Dipartimento di Informatica e
Scienze dell'Informazione

GSPN Analysis in a Distributed Environment

by

Rubén Carvajal-Schiaffino

Theses Series DISI-TH-2001-03

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

GSPN Analysis in a Distributed Environment

by

Rubén Carvajal-Schiaffino

December, 2001

Dottorato di Ricerca in Informatica
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35
I-16146 Genova, Italy
http://www.disi.unige.it/

Ph.D. Thesis in Computer Science

Submitted by

Rubén Carvajal-Schiaffino
DISI, Università degli Studi di Genova, Italy
ruben@disi.unige.it

Date of submission: December 2001

Title:

GSPN Analysis in a Distributed Environment

Advisor:

Prof. Giovanni Chiola
DISI, Università degli Studi di Genova, Italy
chiola@disi.unige.it

Ext. Reviewers:

Prof. Giuliana Franceschinis
Dipartimento di Scienze e Tecnologie Avanzate
Università del Piemonte Orientale "A. Avogadro", Alessandria, Italy
giuliana@unipmn.it

Prof. Jean-Michel Ilié
Laboratoire d'Informatique de Paris 6
Université Pierre et Marie Curie, Paris, France
Jean-Michel.Ilie@lip6.fr

Abstract

Generalized Stochastic Petri Nets (GSPNs) allow one to describe both the logical and temporal evolution of a system in a unified framework. However, the quantitative analysis of GSPN models of "real" systems often generates very large state spaces that are not always solvable with the available tools.

This thesis presents the development of a distributed tool for the construction of the tangible reachability graph, the construction of the infinitesimal generator of the associated Markov chain and its numerical solution.

In particular, a sequential algorithm for the construction of the tangible reachability graph is presented. This algorithm is based on a data structure that encodes canonical firing count vectors starting from the initial marking rather than token distribution. This algorithm applies to bounded and consistent Petri nets models. The main concepts for this new state representation are defined. A comparison with a standard tool in terms of performance is also presented.

The subsequent step is the construction of a distributed version of the reachability graph algorithm using the message passing interface (MPI) approach. A distributed algorithm alleviates the explosion in the size of the underlying state space representation of large GSPN models by partitioning data structures over the memory units of several machines connected by a local area network. The use of the MPI library makes the tool portable on a large class of distributed and parallel architectures in particular on PC cluster. A performance comparison with a standard sequential tool is presented as well.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Performance Evaluation is used to examine a system, to model it and then to try to change the various parameters in the model. Usually the model is used to help increase the effectiveness or efficiency of a system.

There are several paradigms to address Performance Evaluation problems, the most important ones are (divided into formalisms and techniques):

- Formalisms
  - Petri Nets
  - Analytical Queueing Models
  - Process Algebras
- Techniques
  - Benchmarking
  - Discrete Event Simulation
  - Analytical Models (Operational Analysis, Markovian Techniques, etc.)

Petri Nets (PNs) can be used for the formal description of systems whose dynamics are characterized by:

- Concurrency
- Synchronization
- Mutual Exclusion

- Conflicts

which are typical features of distributed environments [3].

PNs incorporate a notion of (distributed) state and rules for state change that allow them to capture both the static and dynamic behaviour of real systems.

In 1962 Carl Adam Petri introduced, in his doctoral thesis [43], PNs in order to describe concurrent systems in terms of cause/effect relations without explicit time considerations. Several years later C. Ramchandani [45], P.M. Merlin and D.J. Faber [37], and J. Sifakis [51], with different approaches, proposed the introduction of temporal concepts into PN models.

Different proposals followed, mainly based on the use of deterministic timing. F. Symons [55], G. Florin and S. Natkin [28], and M. Molloy [40] gave the first definitions of PNs with stochastic time. These proposals opened the possibility of linking PNs to the field of performance evaluation, traditionally based on stochastic modelling. These models and their offsprings are named Stochastic PNs (SPNs). Ajmone, Balbo and Conte [2] proposed an extension of this model where the stochastic timing is mixed with null delays of deterministic type. This model can be used to describe the temporal and logic evolution of a system within a single framework. The resulting modelling paradigm is called *Generalized SPNs* (GSPNs).

1.1 The Problem

GSPNs represent a well-established methodology for performance analysis of concurrent systems. However, the quantitative analysis of GSPNs models of "real" systems often generates state spaces whose size is such that not always it is possible to solve with the available tools.
Although alternative techniques have been developed, the "exact" numerical solution remains the mainstream approach for the quantitative analysis of GSPN models. GSPN numerical solution involves two main steps:

- Enumeration of the state space representation, called **Tangible Reachability Graph (TRG)**.
- Construction and solution of the infinitesimal generator of the associated *Markov chain* (MC).

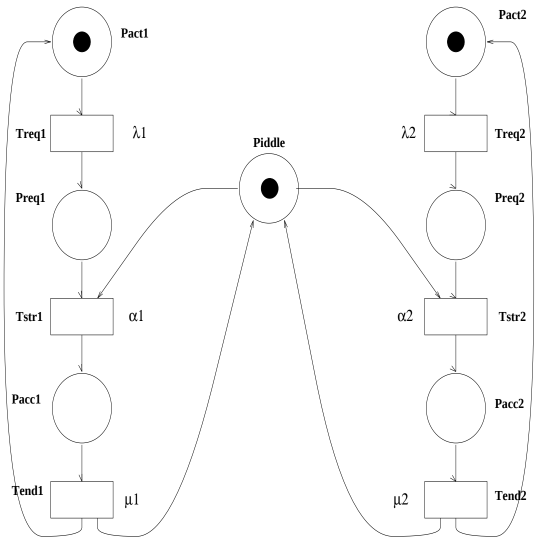


Figure 1.1: SPN model of shared memory system.

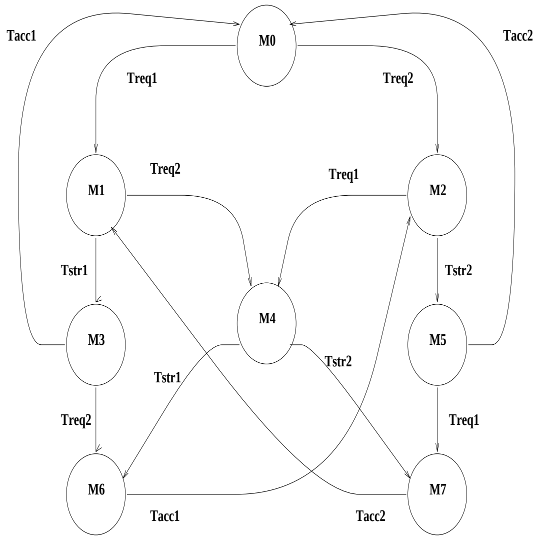
The construction of the TRG presents the state explosion problem, i.e. the size of the graph may grow exponentially with respect to the size of the GSPN model to analize. For this reason the existing sequential tools [44] suffer restrictions in their application due to the limitation of the computational and memory resources.

1.2 State of the Art

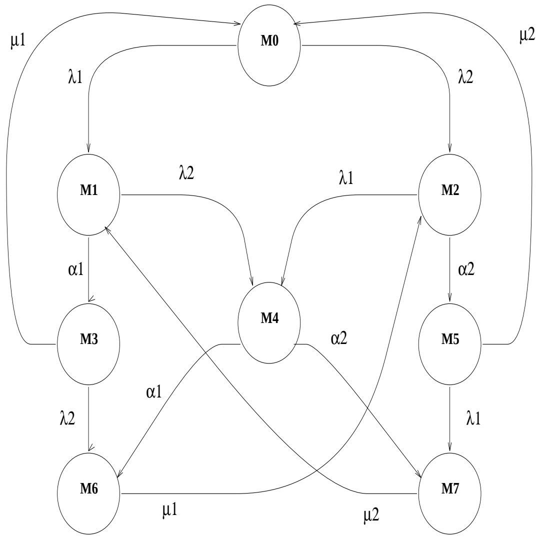
We consider two aspect of the problem: the state explosion problem and marking representation.

Usually the reachability graph of a Petri net system is very large (see e.g. [3, 2]). The memory capacity and performance of a typical processor may therefore impose severe practical restrictions on the set of tractable models. In the literature several approaches have been proposed to deal with the state explosion problem:

- Size reduction methods. They can be further classified in *explicit methods* in which the complete structure of the original "Reachability Graph" (**RG**) is encoded in a way that one can recover it (for instance, in order to generate an exact Markov chain for performance analysis) as opposed to *implicit methods* in which the original structure of the RG is lost, and only some of its original properties are preserved.
  - **Stubborn Sets.** The theory of *stubborn sets* takes advantage of concurrency, or more generally, of the lack of interaction between transitions, captured by the



(a)



(b)

Figure 1.2: (a) The reachability graph of the SPN system of Figure 1.1; (b) The state transition rate diagram of the associated Markov chain.

notion of stubborn sets. This permits the generation of a reduced state space [57] that preserves the liveness of transitions. This approach is an example of an implicit method.

  – **Well-Formed Nets** (WN). The formalism of *WN* [56] is a subclass of *Colored Petri Nets (CPNs)*. State space reduction is obtained in this case by taking the symmetry properties of the model into account. This formalism has proven to be a good modelling tool for complex systems with inherently high degree of symmetry [11]. It can be classified as an explicit method, as the unfolding of the original RG is always possible starting from the Symbolic RG (SRG) representation that is adopted.
  – **Superposed Generalized Stochastic Petri Nets** (SGSPN). An SGSPN is basically a set of GSPNs that synchronize on a common subset of transitions of equal rate [25, 26]. Kronecker algebra is used in order to keep the state transition matrix representation in compact form. Also in this case the method is explicit.
  – **Stochastic Process Algebras** (SPA). *SPA* are modelling formalisms that can be regarded as timed extensions of classical process algebras like CCS and CSP [32]. An extended concept of bisimulation is used to substitute a Markov chain representating with many states by an equivalent lumped Markov chain with fewer states.

- Tolerate the size of the state space
  – Limiting the computation to data relevant for the required output quantities [49]
  – Reducing complexity at the net level [1]
  – Use of disk-based tools [20] to storage large reachability sets (RS) on disks rather than RAM.
  – Parallelization with distribution of the RS representation over several machines with their own memory modules [8, 4, 35].
  – Use of Binary Decision Diagrams to encode sets of markings rather than individual marking representations [39, 41, 48]. This is an example of an implicit method that is difficult to exploit for performance evaluation purposes.

1.3 Contribution of this Thesis

This thesis presents the development of a distributed tool for the construction of the tangible reachability graph, the construction of the infinitesimal generator of the associated

Markov chain and its numerical solution.

In particular, a sequential algorithm for the construction of the tangible reachability graph is presented. This algorithm is based on a data structure that encodes canonical firing count vectors starting from the initial marking rather than token distribution. This algorithm applies to bounded and consistent Petri nets models. The main concepts for this new state representation are defined.

We show that our algorithm generates the same graph that one using an encoding based in the token distribution, also we have found properties of the Reachability Graph that allow us to define efficients data structure with a low overhead. A result of the associated properties with the Reachability Graph is the fact that our algorithm can avoid to search in the whole state space in order to verify the eventual existence of a state. Our algorithm compared with the standard solver GreatSPN TRG builder [12] presents a better performance in space resource and a comparable performance in time resource.

Another contribution of this thesis is the use of our encoding for the validation of bounded Petri Nets. Our method combines *state enumeration* and *structural techniques* in order to compute *under-approximations* of the reachability set and graph of a net. The method is based on *two heuristics* that exploit properties of T-semiflows to detect acyclic behaviors. The resulting algorithm works on a *sliding window* covering a fixed-a-priori number of the last levels of the partially constructed reachability graph. This property allows us to organize the space used to store the reachable markings as a circular array. We apply the method to examples taken from the literature [3, 18, 35]. Our algorithm returns *exact* results in all the experiments. In some examples, the circular array and the sliding window allow us to use 98% less memory space, and to scale up the specification to an initial marking with 255 tokens.

The subsequent step is the construction of a distributed version of the reachability graph algorithm using the message passing interface (MPI) approach. A distributed algorithm alleviates the explosion in the size of the underlying state space representation of large GSPN models by partitioning data structures over the memory units of several machines connected by a local area network. A first contribution of our distributed algorithm is the natural adaptation of the data structures proposed in the sequential version. Moreover, our distributed version of the algorithm showed reasonably good speed on clusters with fast interconnection networks, allowing to further increase the total size of problems that can be solved witout substancial speed penalties.

1.4 About this Thesis

The organization of this thesis is the following:

- Chapter 2 first introduces the main definition and notation about Petri nets. The chapter then presents the classical approach to the problem of the state representation.
- Chapter 3 contains our new state representation and some properties associated with the RG. We also discuss how these properties allow to the implementation of a more space-efficient algorithm. Subsequently we show the results of our experiments.
- Chapter 4 presents the implementation of an algorithm that construct a partial reachability set to verify safety properties for untimed model.
- Chapter 5 presents a distributed version of our algorithm.
- Finally, Chapter 6 presents some conclusions and perspectives for future work on this line.

Chapter 2

Preliminaries

Here we summarize the definitions and notation (from [56]) on which we shall rely throughout the rest of the thesis. The type of nets we focus on are place/transition nets with priorities and inhibitor arcs, as the ones supported by the GreatSPN tool [12]. The idea is that we could substitute the traditional RG builder of the GreatSPN package if we are able to prove that our new algorithm performs better than the original one.

Definition 2.1 A P/T net is a four-tuple $\mathcal{N} = <$ P, T, **Pre**, **Post** $>$ where

- P and T are disjoint finite non-empty set, the places and transition respectively. An arbitrary order of the elements of P and T is implicitly assumed, in order to represent Pre and Post relations in matrix form;
- $\mathbf{Pre}, \mathbf{Post} \in I\!N^{|P|\times|T|}$ are the pre- and post- incidence matrices. The elements of **Pre** represent the capacity weights of the arcs from places to transitions, i.e., the firing conditions for the transitions. The elements of **Post** represent the capacity weights of the arcs from transitions to places. Post(t) is a column vector showing the output upon the firing of a particular transition t.

Definition 2.2 The *token flow matrix* C is a matrix representing the marking change when transitions fire. It is defined as $C = Post - Pre$.

Definition 2.3 A P/T net with priorities and inhibitors arcs is a six-tuple $\mathcal{N} = <$ P, T, Pri, **Pre**, **Post**, **Inh** $>$ where

- $\mathcal{N} = <$ P, T, **Pre**,**Post** $>$ is a P/T net;

- $Pri \in I\!N^{|T|}$ is the priority level associated with each transition (zero by default);
- $\mathbf{Inh} \in I\!N^{|P| \times |T|}$ is the inhibition incidence matrix.

Definition 2.4 A *marking* M is a mapping from places to natural numbers. It is usually represented as a (column) vector of natural numbers of dimension $|P|$ showing tokens assigned to places in $\mathcal{N}$. The order of the entries is determined by the arbitrary order of the elements of P. The initial marking is denoted M_0 .

Definition 2.5 The *characteristic vector* e_t is a column vector of dimension $|T|$ with a single entry 1 in the position corresponding to transition t and 0 elsewhere. With the position of the 1 it simply selects transition t.

Definition 2.6 A transition t is enabled in a marking M (denoted $t \in E(M)$) iff $\forall p \in P$, $M[p] \geq Pre[p, t]$ and $\forall p \in P$ such that $Inh[p, t] > 0$, $M[p] < Inh[p, t]$.

Definition 2.7 A transition $t \in T$ that is enabled in marking M (i.e., $t \in E(M)$) may fire, producing a new marking M'. The firing of t is denoted $M \xrightarrow{t} M'$. The new marking M' is computed according to the following rule.

Definition 2.8 The *state transition equation* of a net $\mathcal{N}$ for transition t is $M_i \xrightarrow{t} M_{i+1}$ such that $M_{i+1} = M_i + C \cdot e_t$ where M_i is the present state, M_{i+1} represents the next state, and $C \cdot e_t$ (denoting the row by column product of matrix C times column vector e_t) is the change in the marking when transition t fires.

Definition 2.9 A *t-invariant* is a (column) vector of size $|T|$ that represents a mapping $\sigma : T \mapsto Z\!\!\!Z$ such that $C \cdot \sigma = \overline{0}$ (where $\overline{0}$ denotes a vector of appropriate size with all entries 0). Any *t-invariant* may be expressed as a linear combination of a finite set of k elements that form a base, where $k = |T| - rank(C)$.

Definition 2.10 A *t-semiflow* is a vector $\sigma : T \mapsto I\!N$ such that $C \cdot \sigma = \overline{0}$. Any *t-semiflow* can be expressed as a weighted sum of a finite set of *minimal support t-semiflows.*

Definition 2.11 A net is called *consistent* if and only if the set of transitions is covered by t-semiflows (i.e., if each transition is included in the support of at least one t-semiflow). Consistency is a necessary condition for a net to be simultaneously live and bounded.

Property 2.1 If a net is consistent the number of minimal support *t-semiflows* is greater than or equal to the number k of elements in a base of *t-invariants*.

Definition 2.12 An occurrence or firing sequence from M is a sequence $\alpha = t_1 \ldots t_k \in T^w$ such that $M \xrightarrow{t_1} M_1 \ldots \xrightarrow{t_k} M_k$. If the firing of sequence α yields marking M', this is denoted by $M \xrightarrow{\alpha} M'$. The (potentially infinite) set of all occurrence sequences of a net system is denoted $L(\mathcal{N}, M_0)$.

Definition 2.13 The firing count vector associated with an occurrence sequence $x \in I\!N^{|T|}$. Given an arbitrary, non-empty sequence $\alpha : M \xrightarrow{\alpha} M'$, its firing count vector $x = \|\alpha\|$ is recursively defined as:

$$\begin{array}{lll} \text{if } \alpha = t & \text{then} & x = e_t \\ \text{otherwise } \alpha = \beta t & \text{then} & x = \|\beta\| + e_t. \end{array}$$

Proposition 2.1 Cycles in the RG and *t-semiflows*. If $\exists M, \exists \alpha \in L(\mathcal{N}, M_0)$ such that $M \xrightarrow{\alpha} M$ then $\exists x$ such that $x = \|\alpha\|$ and $C \cdot x = \overline{0}$. Notice that this property is interesting in the non-trivial case $\|\alpha\| \neq 0$.

2.1 Marking Representation

The first problem to consider in devising a data structure for the reachability graph of a Petri net is the choice of a good coding schema for markings. The obvious choice that was followed in literature so far was to find an encoding for the number of tokens in each place.

Typically two kinds of data structures are used in the construction of the RG: *hashing* [35], and *search trees* [9, 4, 18, 30, 5]. Hashing would work reasonably well if we had a good bound on the final size of the RG. Unfortunately this is not usually the case in practice.

A second approach Chiola [9] proposed to limit the analysis to PN in which all places are *k-bounded*, and used an array of numbers each of which is coded in $\lceil log_2(k+1) \rceil$ bits. Furthermore the following improvements were suggested:

k-bounding for each place of the PN Consider the actual limit k_i on the numbers of tokens that may accumulate in each place p_i of a particular net, assigning exactly $\lceil log_2(k_i+1) \rceil$ bits to its coding in markings. However no algorithm is known to solve this problem in general; for this reason it is often possible to obtain an upper bound for the *place k-bounding* from the analysis of minimal-support *p-semiflows* [19].

Linear Place-Invariants A second improvement in the coding of place markings can be obtained in many cases by observing that *p-semiflows* may be considered as closed paths followed by the tokens when flowing through the net. This strategy can eliminate some redundant places from the coding: a place in which the number of tokens is constant in all reachable markings constitute a place-invariant by itself, therefore the position of its tokens can be coded in $log_2(1) = 0$ bits.

Implicit Places Another kind of redundancy can be eliminated by testing the presence of *implicit* places [19]. A place is said to be implicit when its marking can be obtained as a linear combination (with non-negative integer coefficients) of the marking of a set of other places, that are called implicant. A slightly modified version of the algorithm used to compute place-invariants [52] allows one to eliminate implicit places from the marking to be encoded,and substitute the decoding of their marking with the computation of the linear combination of the implicant.

Non-Linear Place-Invariant and Redundancy A compact coding can be constructed based on a non-linear invariant expression, by encoding separately the position of tokens inside conditional invariants.

The problem of encoding markings in an explicit way has been extensively studied, and a large number of variations have been implemented in various tools. We think that now most of the possible options have been explored, and therefore new improvements can only result from a drastic change of representation methods.

Chapter 3

An Encoding Based on the Firing Count Vectors

Typically the encoding of the states of the RG is based on the tokens distribution among the places of the model (definition 2.4). We have developed an alternative encoding based on the counting of the transitions fired to reach each state starting from M_0. The basic idea is to use the net state equation $M = M_0 + C \cdot \sigma$ and try to encode σ rather than M. We believe that such change in approach may yield several benefits as compared to the standard approach and hope that our prototype implementation may prove the effectiveness of such benefits.

A first problem to consider with this approach is that in case of bounded and consistent nets each reachable marking M' admits infinite solutions to the state equation $M' = M_0 + C \cdot \sigma$. If we want to substitute the firing count vector σ to the marking representation M we have thus to develop criteria for a canonical representation of the (potentially infinitely) many possible ones. We achieve this result using a base of *t-invariants*.

A first step in the definition of a canonical representation of a firing count vector σ consists in requiring that $\forall \xi : C \cdot \xi = \overline{0} \wedge \xi > 0,\ \sigma \not\geq \xi$, i.e., that σ does not contain any *t-semiflow*. For example, the net shown in Figure 3.1 has a base of two *t-invariants* (that happen to be also a minimal set of two *t-semiflows*): $T_1 = [1, 0, 1, 0, 1, 1, 0, 1, 0, 1]$ and $T_2 = [0, 1, 0, 1, 0, 1, 1, 0, 1, 1]$.

Notice that we have considered a version of the net without priority specification. We shall restrict ourselves to the simpler case of nets without priorities, and delay the discussion of the problem of nets with priorities to the last part of the chapter. The results presented in this chapter appeared in [10].

For each M generated we obtain a marking representation as shown in Table 3.1. Notice that the firing count vector representation for all these states, as illustrated in Table 3.1, satisfy the minimality condition. However, in order to correctly point the outgoing arcs from states M_4 and M_5 towards the initial marking representation of M_0, we have to subtract one of the two semi-flows from the "natural" representation of the same marking deriving by the addition of e_6 to σ_4 and of e_7 to σ_5. By reasoning on this example, one might be tempted to conclude that a reachability graph algorithm might be organized as follows.

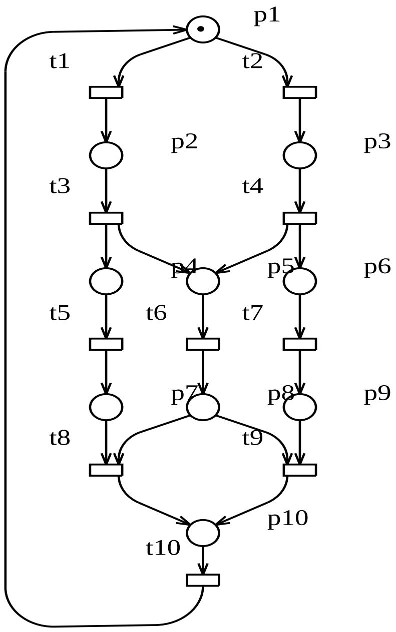


Figure 3.1: A consistent net

"Start with the initial marking M_0 and encode it as the empty firing count vector $\sigma_0^T = [0, \ldots, 0]$. Take non-examined markings $M = M_0 + C \cdot \sigma$ and compute the set of enabled transitions. For all enabled transitions produce the new marking representation $\sigma' = \sigma + e_t$. If $\sigma' \geq$ *t-semiflow* x, reduce $\sigma' \longleftarrow (\sigma' - x)$. Compare σ' with previously stored representations to decide whether the marking $M' = M_0 + C \cdot \sigma'$ is new or not. Repeat until no new marking is found."

Such a simple normalization of σ' is sufficient to reduce the representation of any arbitrary reachable marking to a finite vector. However it may produce redundant state representations, which may decrease the efficiency of the algorithm in terms of memory requirements and/or computation time. This happens when the following condition holds:

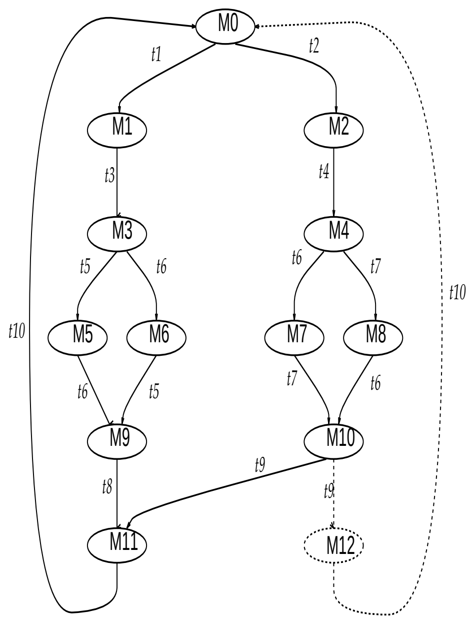


Figure 3.2: The Reachability Graph of the net in Figure 3.1.

Proposition 3.1 If $\exists M \in RS(M_0)$ and $\exists t_a, t_b \in E(M)$ such that there are two minimal support t-semiflows T_a, T_b that verify the condition $T_a \cap T_b \neq \emptyset$ and $e_a \leq T_a \wedge e_b \leq T_b \wedge e_b \not\leq T_a \wedge e_a \not\leq T_b$, then there will be redundant state representations in the RG.

For example, in the net of Figure 3.1:

$$E(M_0) = \{t_1, t_2\}$$

$$T_1 \cap T_2 = \{t_6, t_{10}\}$$

In this case the generated RG using the minimal firing count vector as marking representation would have a redundant state representation as illustrated in Figure 3.2.

A solution to the above problem consists in normalizing with a signed t-invariant $T_c = T_a - T_b$ such that T_a, T_b are the two semiflows involved in the problem. In the example represented in Figure 3.2, if we compute $T_3 = T_2 - T_1$ we obtain: $T_3 = [-1, 1, -1, 1, -1, 0, 1, -1, 1, 0]$. From $\sigma_{10} = [0, 1, 0, 1, 0, 1, 1, 0, 0, 0]$ the enabled transition is t_9 and the new marking is represented by $\sigma' = [0, 1, 0, 1, 0, 1, 1, 0, 1, 0]$, computing $\sigma' - T_3$ we get $\sigma_{11} = [1, 0, 1, 0, 1, 1, 0, 1, 0, 0]$, that is the correct result.

In order to obtain a general solution for this problem, we introduce the concept of a canonical firing vector representation for each marking, based on lexicographic ordering of equivalent vectors. Equivalence classes are defined by linear combinations of *t-invariants*.

State	Place Based Repr.	Transition Based Repr.
0	$M_0 = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0$	$\sigma_0 = 0, 0, 0, 0, 0, 0, 0, 0, 0, 0$
1	$M_1 = 0, 1, 0, 0, 0, 0, 0, 0, 0, 0$	$\sigma_1 = 1, 0, 0, 0, 0, 0, 0, 0, 0, 0$
2	$M_2 = 0, 0, 1, 0, 0, 0, 0, 0, 0, 0$	$\sigma_2 = 0, 1, 0, 0, 0, 0, 0, 0, 0, 0$
3	$M_3 = 0, 0, 0, 1, 1, 0, 0, 0, 0, 0$	$\sigma_3 = 1, 0, 1, 0, 0, 0, 0, 0, 0, 0$
4	$M_4 = 0, 0, 0, 0, 1, 1, 0, 0, 0, 0$	$\sigma_4 = 0, 1, 0, 1, 0, 0, 0, 0, 0, 0$
5	$M_5 = 0, 0, 0, 0, 1, 0, 1, 0, 0, 0$	$\sigma_5 = 1, 0, 1, 0, 1, 0, 0, 0, 0, 0$
6	$M_6 = 0, 0, 0, 1, 0, 0, 0, 1, 0, 0$	$\sigma_6 = 1, 0, 1, 0, 0, 1, 0, 0, 0, 0$
7	$M_7 = 0, 0, 0, 0, 0, 1, 0, 1, 0, 0$	$\sigma_7 = 0, 1, 0, 1, 0, 1, 0, 0, 0, 0$
8	$M_8 = 0, 0, 0, 0, 1, 0, 0, 0, 1, 0$	$\sigma_8 = 0, 1, 0, 1, 0, 0, 1, 0, 0, 0$
9	$M_9 = 0, 0, 0, 0, 0, 0, 1, 1, 0, 0$	$\sigma_9 = 1, 0, 1, 0, 1, 1, 0, 0, 0, 0$
10	$M_{10} = 0, 0, 0, 0, 0, 0, 0, 1, 1, 0$	$\sigma_{10} = 0, 1, 0, 1, 0, 1, 1, 0, 0, 0$
11	$M_{11} = 0, 0, 0, 0, 0, 0, 0, 0, 0, 1$	$\sigma_{11} = 1, 0, 1, 0, 1, 1, 0, 1, 0, 0$

Table 3.1: Marking Encoding for the Reachability Graph of Figure 3.2.

Definition 3.1 Two firing count vectors σ and σ' are equivalent (denoted $\sigma \approx \sigma'$) iff $\sigma = \sigma' + \sum_{i=1}^{k} a_i \cdot x_i$ where the x_i are the k elements of a base of *t-invariants* and $a_i \in \mathbb{Z}$.

Definition 3.2 A firing count vector σ is lexicographically less than or equal to another firing count vector σ' (denoted $\sigma \preceq \sigma'$) iff $\exists i \geq 0$ such that $\forall j \leq i$, $\sigma[j] = \sigma'[j] \wedge (i = |T| \vee \sigma[i] < \sigma'[i])$.

Definition 3.4 A firing count vector σ is lexicographically minimal (also called canonical) iff $\forall \sigma' \approx \sigma$, $\sigma \preceq \sigma'$.

A final improvement of the proposed marking encoding is thus achieved using linear combinations of *t-invariants* in order to compute canonical representations of the firing count vector. A possible disadvantage of this approach is the overhead of linear combinations computations. Such overhead can be reduced by caching already computed linear combinations in order to avoid their subsequent re-computations. Particularly useful linear combinations are *t-semiflows*, that can be added to the cache right from the beginning, as they represent the closure of cycles in the RG, and therefore will probably be used during the generation of the RG itself. Such a “pre-fetching” of the cache, of course, might not be practical in cases in which the set of minimal t-semiflows is very large. In any case, the algorithm that we present requires only the pre-computation of a base of t-invariants, which can always be computed in polynomial time in the number of places and transitions.

3.1 The New RG Algorithm

The proposed marking encoding imposes an additional step to the RG generation algorithm that consists in the normalization of the firing count vector to make it canonical. This is always possible for bounded and consistent models. It is still unclear to us whether our new algorithm might be applicable to inconsistent models (while it does not make sense to start the RG construction for unbounded nets).

Our normalization algorithm is based on a pre-computation of a base of t-invariant, that can be computed in polynomial time as a function of the size of the net. The algorithm is presented in a simplified way compared to the actual implementation in our prototype. Possible optimizations includes the possibility of adding minimal-support t-semiflows to the initial list of invariants (in case the number of semiflows is not very high), and classifying the vectors that are elements of the list of invariants in two subsets: the subset of vectors without negative entries, that are also minimal-support t-semiflows, and the others that contain both strictly positive and negative entries. Moreover, the list of invariants, in practice, may be implemented as an LRU cache of predefined size. The algorithm can be sketched as shown in Figure 3.3.

Notice that function "Normalize" does not need to iterate the subtraction of t-invariants, as it is called on a representation that, by construction, can only be *partially not canonical*. Indeed the initial representation is canonical by assumption, and by adding one single e_t vector we can produce a vector in which the applicability condition of the normalization subtraction can only be satisfied once.

Notice also that the presence or absence of priorities and inhibitor arcs is irrelevant for our algorithm. Of course the presence of priorities and inhibitor arcs must be taken into account for the enabling test of transitions, but it does not change the structure of the algorithm or the coding of the marking in case a transition is found to be enabled under more restrictive conditions.

3.1.1 Some Properties of the RG

The key factor for the efficiency of the implementation is the marking encoding. For this reason we use properties of the RG in order to obtain a competitive marking encoding.

Let us present some useful properties deriving from the adoption of our proposed marking encoding for the construction of the RG:

```
Algorithm ReachG(𝒩, M_0, T_i)
𝒩: a bounded P/T net with 𝒩 =< P, T, Pre, Post, Inh >
M_0: Initial Marking;
T_i: the base of t-invariant;
σ ← [0, 0, ..., 0]: current firing count vector;
NM ← [σ]: an ordered list of markings;
RS ← [σ]: reachability set of the net;
RG ← ∅: reachability graph of the net;
List_i ← T_i;
while ( NM ≠ ∅ )
    let σ be the first element of NM;
    remove σ from NM;
    M ← M_0 + C · σ;
    for ( each transition t_i enabled in M )
        σ' ← σ;
        σ'[i] ← σ'[i] + 1;
        σ' ← Normalize(σ', List_i, T_i);
        if ( σ' ∉ RS )
            insert σ' at the end of NM;
            RS ← RS ∪ σ';
        endif
        RG ← RG ∪ (σ --t_i--> σ');
    endfor
endwhile

Function Normalize(σ',List_i,T_i)
choose some x ∈ List_i such that σ' − x ≥ 0 ∧ σ' ≻ (σ' − x);
if (∃x)
    return (σ' − x);
else
    generate a new t-invariant ni as linear combination of T_i
    such that (σ' − ni) ≥ 0 ∧ σ' ≻ (σ' − ni);
    if (∃ni)
        List_i ← List_i ∪ ni;
        return (σ' − ni);
    else
        return σ';
    endif
endif
```

Figure 3.3: The New RG Algorithm.

Definition of Level. Let $\sigma = [\sigma_1, \sigma_2, \ldots, \sigma_t]$ be a marking associated with the firing count vector, if $\sum_{i=1}^{t} \sigma_i = L(\sigma)$, then σ is defined to be at level L of the RG.

Property 3.1 The level $L(\sigma)$ is completely defined for all canonical σ even in case of multiple paths connecting M_0 to M in the RG.

Definition 3.5. T-invariant consistent with lexicographic order. Consider *t-invariant* $X = X^+ - X^-$ where both X^+ and X^- components are non-negative. X is consistent if $\sum_{i=1}^{|T|} x_i{}^+ = \sum_{i=1}^{|T|} x_i{}^-$. In case $\sum_{i=1}^{|T|} x_i{}^+ > \sum_{i=1}^{|T|} x_i{}^-$ X is lexicographically consistent iff $\forall i : x_i^- > 0$, $\exists j < i : x_j^+ > 0$. In case $\sum_{i=1}^{|T|} x_i{}^+ < \sum_{i=1}^{|T|} x_i{}^-$ X is lexicographically consistent iff $\forall i : x_i^+ > 0$, $\exists j < i : x_j^- > 0$.

Property 3.2 Let us consider a net with transition ordering consistent with all *t-invariants*. Let σ be a marking representation. The markings representation σ_t generated from σ by firing a transition t is in the immediately next level if the normalization does not generate an equivalent marking such that $|\sigma| \leq L$, being L its level in the RG.

Of course this property holds only for the complete RG in case of nets with priorities, and does not generally hold if we encode only the tangible part of the RG in case of nets with GSPN-type priority structure.

Moreover, Property 3.2 applies only to nets in which it is possible to order transitions in such a way that all *t-invariants* with positive and negative signs are consistent with lexicographic ordering. This condition implies that a reduction of lexicographic value by linear combination of *t-invariants* does not increase the level $L(\sigma)$ of the marking representation. In case of nets with priority structure, this also implies that the chosen lexicographic order must be consistent with the priority structure as well.

Property 3.3 Let σ be a marking representation belonging to a given level L. Let σ_1, σ_2, $\ldots$,σ_j be marking representations generated starting from σ such that they belong to level $L+1$. Then, if we choose to fire the enabled transitions in decreasing lexicographical order then, lexicographically $\sigma_1 \succ \sigma_2 \succ \ldots \succ \sigma_j$. Conversely if we choose to fire the enabled transitions in increasing lexicographical order then, lexicographically $\sigma_1 \prec \sigma_2 \prec \ldots \prec \sigma_j$.

Definition 3.6. Depth of the RG The Depth of the RG is given for the maximum level in the RG.

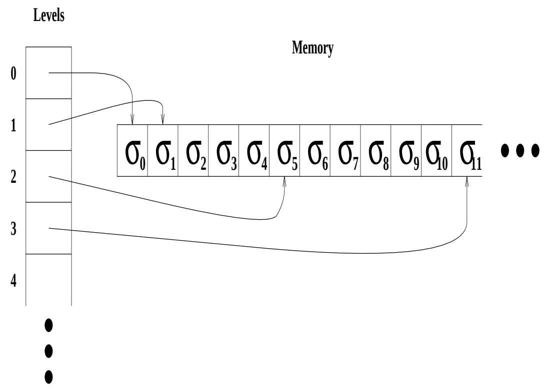


Figure 3.4: The Memory Organization.

Theorem 3.4 The depth of the RG associated with a PN model with parameter K, defined as $Depth(K)$ is given by:

$$Depth(K) \leq \begin{cases} L, \text{ if } K = 1 \\ L * K, \text{ if } K > 1 \end{cases}$$

3.1.2 Efficient Data Structures

Property 3.1 allows us to define a search key in a subset of markings (those that are in the same level). Property 3.3 allows an efficient binary search within each level.

Property 3.2 guarantees that if the new marking representation of σ' is not found at level $L+1$ then the marking was already generated at a lower level. If we are interested only in the reachability set construction we know that in that case the marking is old and we can avoid any further search. If we want to compute the RG, instead, we have to search for the marking any way in order to properly label the arc.

The main advantage of using the named properties is the low overhead to store the markings. Indeed we only need to keep the address of the first marking of each level.

We structure the available memory (RAM + swap area) as an array, where each slot in the array is made of m cells and stores a marking ($m = |T|$), as depicted in Figure 3.4. If we use cells of 8 bits (1 byte), we limit the total count of each transition firing to a maximum of 255. If TM is the size of allocated memory in bytes, the number of available slots AS is $AS = TM/m$. A table contains the initial address of the set of markings of each level, with this value we can perform an efficient search in the respective level in order to determine the existence of a marking.

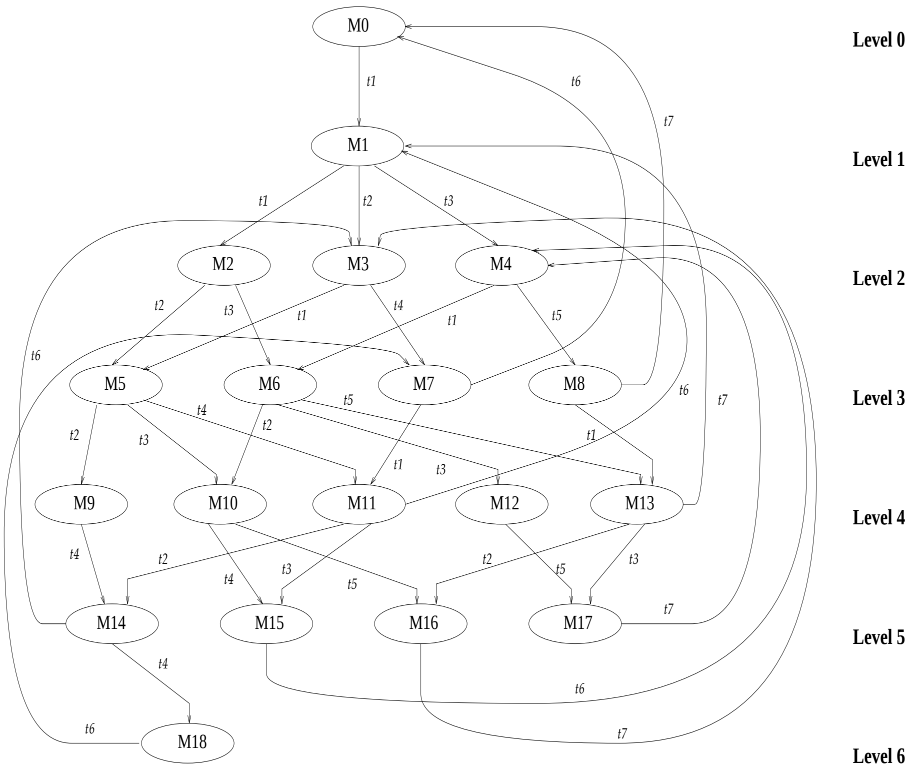


Figure 3.5: The Reachability Graph of Readers & Writers PN System with K = 2.

3.2 Experimental Results

We implemented our algorithm in the C language making sure that it could take as input the network description files adopted by the GreatSPN package. Then we experimentally validated our prototype based on a number of actual examples. We verified first of all the correctness of our algorithm by checking that it produced the same reachability graph structure as the original RG builder of the GreatSPN package. Subsequently we started to compare the performance of the two RG builders both in terms of CPU time and of memory requirements on the same machine.

We believe that with the current trend of low-cost computer technology evolution the main limitation is the memory requirement, that is limiting the maximum size of the RG that can be generated, rather than the CPU time to complete the execution in the cases in which the graph is small enough to fit memory requirements. We run our experiments on two different PC platforms with similar Linux operating system, in order to study the effect of CPU speed and memory size: a) a PC with a Pentium 133 MHz CPU, 32 MB of RAM and 32 MB of swapping area; b) a PC with Athlon 800 MHz CPU, 256 MB of RAM and 256 MB of swapping area.

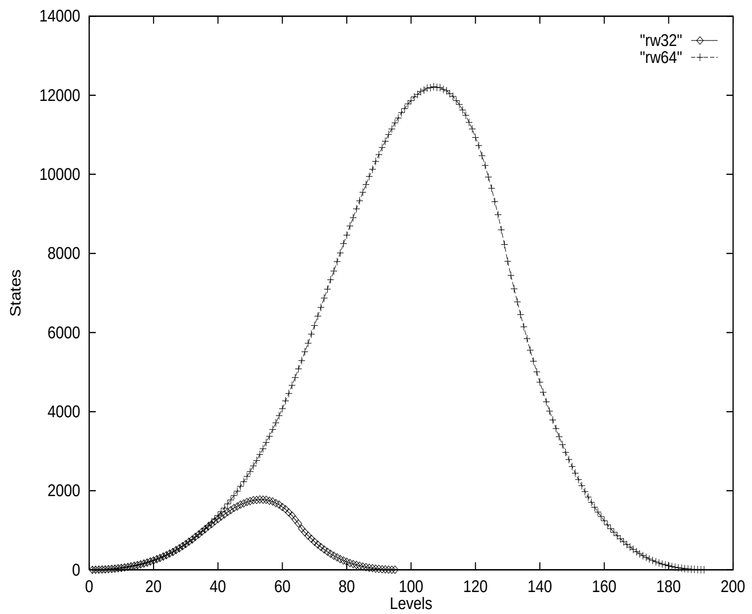


Figure 3.6: Amount of Levels for the Readers & Writers System with K = 32 and K = 64.

For our experiments we have chosen the following PN scalable models taken from various sources of literature, in order to construct a significant set of benchmarks. All considered models happened to be live and bounded (hence consistent). Moreover, it was always possible to find an ordering of transitions so that property B holds true.

A - Readers & Writers System with priorities [3];

B - modified version of the Central Server Model [3], obtained by making all transitions timed;

C - original version of the Central Server Model [3] (with priority specification);

D - modified version of the Kanban Model with Failures and Repairs [3] in the particular case of 4 stages, obtained by making all transitions timed;

E - original version of the Kanban Model with Failures and Repairs [3] in the particular case of 4 stages (with priority specification);

F - Multipoll System [35] (with priority specification).

In all cases we ran the models from low values of their marking parameter K until both GreatSPN and our new prototype could not complete the task because of lack of memory. We present our results in two tables, dividing models containing only timed transitions from models containing immediate transitions as well.

PN Model	K	$\|RG\|$	a) 133 MHz 32+32 Mb		b) 800 MHz 256+256 Mb	
			GreatSPN CPU time	Our program CPU time	GreatSPN CPU time	Our Program CPU time
B (Central Server)	16	13,124	2.16	2.88	0.22	0.37
	32	95,876	27.92	23.18	2.94	2.91
	64	732,420	296.54	191.56	34.09	23.89
	75	1,170,704	538.45	311.53	62.43	38.73
	115	4,162,544	n.c.	1156.24	322.22	143.14
	128	5,724,675	n.c.	n.c.	n.c.	198.78
	240	37,327,044	n.c.	n.c.	n.c.	1,410.12
D (Kanban)	1	4,096	0.74	2.35	0.08	0.30
	2	104,976	25.34	71.02	3.05	8.98
	3	1,048,576	n.c.	794.38	43.58	98.61
	4	6,259,000	n.c.	n.c.	n.c.	645.56

Table 3.2: Results for models without priorities.

Let us first comment on the results reported in Table 3.2, for models B and D without priorities. The computation time of our new algorithm is slightly higher compared to the one of GreatSPN in case of small RGs, probably because of the additional normalization overhead for the marking representation. The exploitation of the properties of the marking representation, however, allowed us to encode the representation in a much more space-efficient way compared against GreatSPN, even if GreatSPN includes sophisticated marking encoding techniques based on the pre-computation of p-semiflows. This better exploitation of memory resources can be appreciated in two ways: first, as in the case of model B, it may yield lower CPU time for larger graphs because of the reduced swapping overhead; second, as can be seen in both examples, it may allow the generation of substantially larger graphs (up to one order of magnitude) using the same overall virtual memory resources. Notice that these considerations apply to both hardware platforms.

The comparison in case of nets with priority is less conclusive. Consider the results reported in Table 3.3 for models A, C, E, and F. The only clear case is the one of model A, where the two algorithms actually produce the same graph. In this case similar remarks as for the results in Table 3.2 may be stated.

In the other 3 cases the comparison is not obvious because the GreatSPN RG builder actually computes only the tangible reachability graph, and does not generate all vanishing markings. Independent subnets of immediate transitions are reduced separately in GreatSPN, so that the number of vanishing markings that have to be stored and searched for is, in some cases, orders of magnitude lower than the actual number of vanishing markings. Our current prototype does not exploit such a partition of the net (although adding this feature is just a matter of programming effort) so that it actually has to generate and store

PN Model					a) 133 MHz 32+32 Mb		b) 800 MHz 256+256 Mb	
			GreatSPN	Our Program	GreatSPN	Our Program	GreatSPN	Our Program
	K	$\|\|TRG\|\|$	$\|\|VRS\|\|$	$\|\|VRS\|\|$	CPU time	CPU time	CPU time	CPU time
A	32	1,057	6,511	6,511	0.83	0.51	0.10	0.06
(Read/Write)	64	4,161	47,839	47,839	7.79	3.24	0.97	0.41
	103	10,713	192,815	192,815	40.62	12.64	6.16	1.77
	128	16,513	366,015	366,015	n.c.	23.97	14.44	3.09
	205	42,231	1,478,049	1,478,049	n.c.	96.78	120.91	13.49
	255	65,281	n.c.	2,828,799	n.c.	184.23	n.c.	23.66
	585	342,811	n.c.	33,709,649	n.c.	n.c.	n.c.	359.59
C (Central Server)	32	1,650	193	4,721	0.35	0.73	0.04	0.11
	64	6,370	385	18,657	1.76	3.53	0.19	0.47
	128	25,026	769	74,177	14.54	14.88	1.52	1.97
	255	98,432	1,531	293,506	99.47	64.15	13.00	8.31
	512	395,011	n.c.	1,181,441	n.c.	402.08	n.c.	50.45
	1540	3,562,793	n.c.	10,677,591	n.c.	n.c.	n.c.	833.24
E (Kanban)	4	8,784	3,578	53,099	2.66	29.78	0.29	3.76
	8	59,024	27,934	384,036	20.07	259.00	2.55	32.41
	14	292,784	148,768	1,963,398	112.84	1,795.90	15.77	219.49
	16	431,736	221,894	2,907,424	n.c.	n.c.	27.93	356.24
	27	1,996,720	1,064,384	13,668,870	n.c.	n.c.	186.63	2,847.51
	31	2,963,356	n.c.	20,243,074	n.c.	n.c.	n.c.	5,050.49
F (Multipoll)	4	12,405	3,125	32,625	4.48	14.46	0.52	1.75
	8	171,765	32,805	407,025	73.40	209.85	9.72	25.70
	9	270,160	50,000	633,000	118.18	334.06	16.33	40.29
	12	822,133	142,805	1,883,505	n.c.	1,051.33	58.30	127.27
	16	2,522,613	417,605	5,683,185	n.c.	n.c.	233.18	410.06
	20	6,047,349	n.c.	13,487,985	n.c.	n.c.	n.c.	1,050.00

Table 3.3: Results for models with priorities.

a substantially higher number of markings to terminate. This consideration may explain the much worse computation time of our prototype, and the fact that in some cases it is not able to produce a reachability graph with a larger number of tangible markings. Nevertheless, in all experiments made we never found a case in which the reduction in the number of vanishing markings operated by GreatSPN could allow GreatSPN to generate larger graphs than our prototype. We believe that the only reasonable conclusion that can be drawn from these experiments is that it could be worthwhile to extend our prototype in order to exploit the same net partition that is currently exploited by GreatSPN, and then repeat the comparisons.

The theorem 3.4 allows us to verify in the middle of the execution whether it will be possible to construct the whole RG or not. If we reach level $K * |T_s|/2$ while still having at least 50% of the memory available, then we know that we can conclude our computation.

Chapter 4

Validation of Bounded Petri Nets

4.1 Introduction

As an interesting application of the encoding based on the *firing count vectors*, in this chapter we will show how it is possible to apply heuristics based on *t-invariants* in order to efficiently explore the reachability set of a bounded Petri Net and to verify its safety properties like mutual exclusion.

Bounded Petri Nets (PNs) are finite-state concurrent systems in which the maximal number of processes (tokens) in any possible state (place) is bounded by a constant. Though decidable, the verification of safety and liveness properties of bounded PNs is a very hard problem in practice. Following the literature in the field [53, 58], we distinguish the techniques used to attack this problem into the following classes.

State Enumeration Techniques. The *reachability graph* of a finite-state system built using an *exhaustive search* algorithm [33] is a complete tool for the *verification* of safety and liveness properties. This technique suffers from the *state explosion problem*, i.e., the explosion of the size of the reachability graph compared to the size of the specification [6, 58]. *Partial search* [33] can be used as heuristics to *validate* large finite-state systems. In general, partial search returns under-approximations of the reachability graph. It cannot be used for verification purposes but only for *simulation* and *testing*. When incorporated in search algorithms, efficient data structures like *hash tables* [33], *BDDs* [6], and *Sharing Trees* [30] represent other important *heuristics* to alleviate state explosion.

```
ALGORITHM ES(N,P) : Boolean.
N = ⟨P,T,Pre,Post,m_0⟩ : PN;
P: the safety property;

Old:=∅;
New:={m_0};
while (New nonempty) do
   m = element from New;
   if not(P(m)) then return(false);
   for every t_i ∈ T enabled at m do
        m' = m + C[P,t_i];
        if (m' ∉ Old ∪ New) then
             add m' to New;
   endf;
   add m to Old;
   delete m from New;
endw;
return(true).
```

```
ALGORITHM DES(N,P) : Boolean.
N = ⟨P,T,Pre,Post,m_0⟩ : PN;
P: the safety property;
y_0 := 0;
OLD := ∅;
NEW := {y_0};
while (NEW nonempty) do
   y = element from NEW;
   if not(P(M(y))) then return(false);
   for every t_i ∈ T enabled at M(y) do
        y' = y[y_i := y_i + 1];
        if (M(y') ∉ M(OLD ∪ NEW)) then
             add y' to NEW;
   endf;
   add y to OLD;
   delete y from NEW;
endw;
return(true).
```

Figure 4.1: Two formulations of the **Type 1** Reachability Algorithm for PNs.

Structural Techniques. While state enumeration is a general-purpose technique for validation of finite-state systems, verification techniques based on *structural properties* are a distinguishing feature of PNs [53]. These techniques work without explicitly computing the reachability graph. They rely on *linear programming* (synonymous of *efficiency*) usually returning *approximated* answers. For instance, the *state equation* [46] can be used to over-approximate the reachability set of a PN, and thus to verify safety properties [53]. Other techniques like *traps* can be used to improve the precision of the state equation [27].

In contrast with traditional uses of structural techniques , in this chapter we investigate their *combination* with *enumerative* techniques for *validating* and *debugging* systems modeled as bounded PNs. Specifically, we use structural properties as *heuristics* to guide the *search* during state exploration. In order to attack state explosion we incorporate our heuristics within a *partial search* algorithm, and we leave open the possibility of using efficient data structures for storing intermediate results. The results presented in this chapter appeared in [7].

4.2 Towards the Combination with State Enumeration

Our starting point consists in a reformulation of the standard exhaustive search algorithm using *firing count vectors* presented in Chapter 3. The unique goal of this preliminary step is to simplify the integration of our *structural* heuristics in the enumerative approach.

4.2.1 Sufficient Conditions for Detecting Acyclic Behaviors

The contraposed form of the next corollary:

Corollary 4.1 (Cycle $\Rightarrow$ T-semiflow [53]). Let N be a PN, and let $m \xrightarrow{\sigma} m'$. If $m = m'$, i.e., σ is a cycle in $\mathcal{G}(N, m)$, then $p_\sigma \in Lin_{\mathbf{Q}^+}(\mathcal{F})$.

As shown next, the *contraposed* form of Cor. 1 can be used to devise *sufficient conditions* for detecting acyclic occurrence sequences without having to search for visited markings.

Corollary 4.2 (Not-T-semiflow $\Rightarrow$ Not-Cycle). Let N be a PN, and let $m \xrightarrow{\sigma} m'$. If $p_\sigma \notin Lin_{\mathbf{Q}^+}(\mathcal{F})$ then σ is a not a cycle in $\mathcal{G}(N, m)$.

This property goes well together with our formulation of the forward reachability algorithm using Firing Count vectors. Before entering in more details, let us remark that to check the condition $p_\sigma \notin Lin_{\mathbf{Q}^+}(\mathcal{F})$ of Cor. 4.2 we must solve a *linear problem* with *rational* solutions (polynomial in the size of $\mathcal{F}$). Are there more efficient sufficient conditions (e.g. linear in $\mathcal{F}$) we can use? To answer this question, let us define the following *new* notion.
Definition 4.1 (Integral Fundamental Set). We say that the fundamental set $\mathcal{F}$ is *integral* whenever $x \in Lin_{\mathbf{Q}^+}(\mathcal{F})$ implies that $x \in Lin_{\mathbf{Z}^+}(\mathcal{F})$, i.e. all T-semiflows can be computed using non-negative combinations with integer coefficients.

Under the assumption that $\mathcal{F}$ is integral, the following theorem can be used as a sufficient condition for detecting acyclic behaviors.

Theorem 4.1 (Sufficient Condition for Not-T-semiflow). Let N be a PN with m transitions, and *integral* fundamental set $\mathcal{F} = \{x_1, \ldots, x_k\}$, where $x_i = \langle x_{i,1}, \ldots, x_{i,m} \rangle$. Furthermore, let $m \xrightarrow{\sigma} m'$, and $p_\sigma = \langle y_1, \ldots, y_m \rangle$ be the *firing count vector* associated to σ. If for all $i : 1, \ldots, k$ there exists $j \in \{i, \ldots, m\}$ such that $x_{i,j} > y_j$, then for all *non-empty* subpath σ' of σ, $p'_\sigma \notin Lin_{\mathbf{Q}^+}(\mathcal{F})$.

The cost of checking the condition of Theorem 4.1 is *linear* in the cardinality of $\mathcal{F}$. The cardinality of $\mathcal{F}$ is potentially *exponential* in the size of N, but it is often linear in practice (see Section 4.4). As a remark, note the difference between the hypothesis of Theorem 4.1, and those of Prop $\mathbf{C}\cdot p_\sigma \neq 0$. If Theorem 4.1 holds, then *all subpaths* contained in the path σ from m to m' are acyclic. Contrary, if $\mathbf{C}\cdot p_\sigma \neq 0$, then we deduce that *only* the paths from m to m' are acyclic. Clearly, there exists a marking m'' such that $m \stackrel{\sigma_2}{\to} m'' \stackrel{\sigma_2}{\to} m'$, $\mathbf{C}\cdot p_{\sigma_1+\sigma_2} \neq 0$, and $\mathbf{C}\cdot p_{\sigma_2} = 0$.

4.2.2 Checking the integrality of $\mathcal{F}$

To check the integrality of the fundamental set, we can use the notion of *total unimodularity* [50]. A matrix $\mathbf{A}$ with integer coefficients is totally unimodular if every subdeterminant of $\mathbf{A}$ is 0, 1 or -1. The extreme points of the set of solutions of $\mathbf{A}\cdot x = b$, $\mathbf{A}$ being totally unimodular, are integer numbers for any vector b [50]. To check the total unimodularity of $\mathcal{F}$, we can use the following (polynomial-time) criterion on the matrix with minimal T-semiflows as *rows*.

Theorem 4.2 (From [50]). Let $\mathbf{A}$ be matrix with two non-zero coefficient in each column. $\mathbf{A}$ is totally unimodular iff its rows can be split into *two classes* such that for each column: if the nonzero in the column have the same sign then they are in different classes, and if they have opposite signs they are both in the same class.

Thus, if $\mathcal{F}$ forms a totally unimodular matrix, then $x \in Lin_{\mathbf{Q}^+}(\mathcal{F})$ if and only if $x \in Lin_{\mathbf{Z}^+}(\mathcal{F})$. Perhaps surprisingly, several examples taken from the literature satisfy the integrality requirement on $\mathcal{F}$. We will turn back to this point in Section 4.4.

4.3 Partial Search with Structural Heuristics

We come now to the definition of our partial search algorithm. Basically, the idea is to replace the core of the reachability algorithm DES of Figure 4.1 with two heuristics selected on the basis of a preliminary comparison of *firing count vectors* with minimal T-semiflows. The first heuristics exploits Theorem 4.1 to add markings to the set of visited states. The second heuristics uses sufficient conditions to *localize* the search for back-edges in the reachability graph. A Boolean flag, (called **complete**), is used to estimate the *quality* of the approximation computed by the resulting (partial search) algorithm. PS (*partial search*) Figure 4.2. To explain it in detail, in the rest of the section we will use the predicate SFC defined as

$$\mathrm{SFC}(y) \doteq \text{for all } x = \langle x_1, \ldots, x_m\rangle \in \mathcal{F} \text{ exists } i \in \{i, \ldots, m\} \text{ s.t. } y_i < x_i.$$

to denote the comparison between a *firing count vector* $y = \langle y_1, \ldots, y_m \rangle$ and the minimal T-semiflows of $\mathcal{F}$. Now, let y' be the new Firing Count vector generated during the execution of forward reachability, and let OLD and NEW denote the set of visited markings.

The First Structural Heuristics. Suppose $\mathrm{SFC}(y')$ holds. From Theorem 4.1 and Cor. 4.2, we can deduce that the marking $M(y')$ is not present in *all paths* σ, $p_\sigma = y'$, going from m_0 to $M(y')$. Under this hypothesis, our heuristics is defined as follows: without further checks on OLD we instruct the algorithm to immediately add y' to NEW. The advantage of the heuristics is that we avoid the cost of searching for (a possible occurrence of) $M(y')$ in the whole graph. The drawback is that it could introduce redundant markings. In fact, the marking $M(y')$ may occur in paths unrelated to y' (not captured by Cor. 4.2). This fact does not influence the termination of the resulting algorithm, as stated in Theorem 4.2. We postpone the practical evaluation of the first heuristics to Section 4.4.

The Second Structural Heuristics. Suppose that $\mathrm{SFC}(y')$ does not hold. Then, there exists some $x \in \mathcal{F}$ such that $y' \geq x$. In other words, all paths σ such that $p_\sigma = y'$ contain a subpath that is a minimal T-semiflow. Furthermore, since by definition $\mathbf{C} \cdot x = 0$, if we apply the state equation we obtain that

$$M(y') \;=\; m_0 + \mathbf{C} \cdot (y' - x).$$

Our idea is to use the *normalized firing count vector* $y' - x$ to guide the search for a marking $m \in$ OLD such that $m = M(y')$.

Then, if $\mathrm{SFC}(y') = false$, we first search for a marking m such that $m = M(y')$ in all levels $\mathrm{OLD}[Level(y' - x)]$ with $x \in \mathcal{F}$ and $y' \geq x$. If we find the node we draw a back-edge. The edge will be part of a cycle. If the previous *local* search fails, we discharge the vector y' and set the Boolean flag **complete** to *false* to inform the user that the algorithm is computing an *under-approximation* of the reachability graph. Basically, we substitute the *full termination test* $y \in$ OLD of the algorithm DES of Figure 4.1 with a sufficient condition.

If the flag **complete** is true when terminates the exploration of the state space, then the resulting reachability set is exact. The following theorem formalizes these properties.

Theorem 4.3. Let N be a bounded PN with *integral* fundamental set $\mathcal{F}$, $\mathcal{P}$ a safety property and let C be the value of the flag **complete** when the algorithm PS of Figure 4.2 returns. Then,

(i) $\mathrm{PS}(N, \mathcal{P})$ always terminates (returning $true$ or $false$);
(ii) if $\mathrm{PS}(N, \mathcal{P}) = true$ and $C = true$, then $\mathcal{P}$ holds for N;
(iii) if $\mathrm{PS}(N, \mathcal{P}) = false$, then $\mathcal{P}$ does not hold for N.

ALGORITHM PS($N, \mathcal{P}$): Boolean
$N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$;
$\mathcal{F}$: *integral* fundamental set of N;
$\mathcal{P}$: the *safety* property;

```
y0 := 0; complete:=true;
NEW := {y0}; OLD := ∅;
while (NEW nonempty) do
    y = element from NEW;
    if not(P(M(y))) then return(false);
    for every t_i ∈ T enabled at M(y) do
      y' = y[y_i := y_i + 1];
        if for all x ∈ F exist i ∈ {1,...,m} s.t. y'_i − x_i < 0 then
                if (M(y') is not in M(NEW)) then add y' to NEW;
        else if (M(y') is not in M(Old[rank(y' − x)]) for some x ∈ F, y' ≥ x) then
                complete:=false;
             else add the back-edge;
    endfor;
    add y to OLD; delete y from NEW;
endw;
if complete write('Exact RS') else write('Approximated RS');
return(true).
```

Figure 4.2: A **Type 2** Reachability Algorithm.

The second heuristics gives us a bound on the number of levels we have to keep in memory during the exploration of the reachability graph. The bound WS (window size) is the maximum between the *levels* of the minimal T-semiflows in $\mathcal{F}$, namely

$$WS \ = \ max\{ \ Level(x) \mid x \in \mathcal{F} \ \}.$$

Thus, our algorithm works only on a *window* of dimension WS that covers the last levels of the current reachability set. We will present a memory management based on this property in the next section.

4.4 Experimental Results

We have implemented a prototype version of the algorithm DFR of Figure 4.2, borrowing the graphical interface and the library for computing structural properties from GreatSPN [12], and using the following specialized memory management.

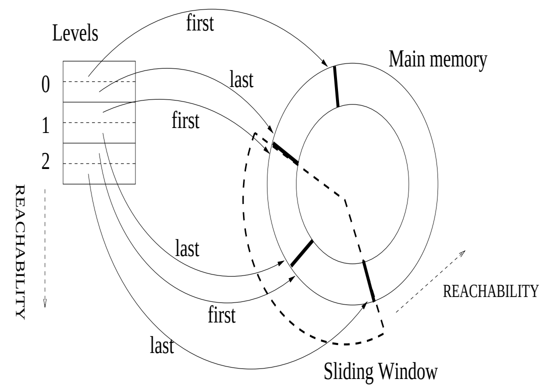


Figure 4.3: The Memory Organization.

4.4.1 Organizing the Memory as a Circular Array

We consider PNs where transitions can be fired at most 255 times. We organize the available memory (RAM + swap area) as a *circular array*, where each slot in the array contains m bytes and stores a *firing count vector* (m=number of transitions). Our representation does not depend on the bound on the number of tokens in the places. If TM is the size of allocated memory in bytes, the number of available slots AS is then $AS = TM/m$. Furthermore, if NS is the number or reachable vectors, then the memory required to store them is $MS = NS * m$ bytes. A table maintains the initial and final address of the set of *firing count vectors* of each level; each level is an ordered list. A *sliding window* covering the last WS levels of the reachability graph moves around the circular array. The global size of the sliding window is the sum of the number of states in each of its WS levels. By construction of PS, all states outside the window will never be considered again in successive iterations. Thus, the memory allocated for them can always be re-used. An *overflow exception* OF is raised as soon as the algorithm adds a slot of the last level of the window to its first level (i.e. the window covers all memory). NR will indicates the number of times the *last* slot of the sliding window goes beyond the rightmost limit of the array ($NR = 0$ means $MS \leq TM$), and $VM = Ns * m$ will indicate the *virtual memory* need to store the entire reachability set. The ratio $R = TM/VM$ indicates our gain in terms of memory occupancy.

4.4.2 Practical Evaluation

At this stage of our work, the purposes of the experiments were: (1) testing the *applicability* of the assumptions under which the algorithm works (the existence of an integral fundamental set); (2) testing the *quality* and *efficiency* of our heuristics; (3) testing the *scalability* of the specialized memory management.

CASE-STUDY	T	P	SF	ET	WS	I?
Kanban [18]	16	16	5	0.01s	8	$\surd$
Flexible Manufacturing System (FMS) [18]	20	22	4	0.04s	13	$\surd$
Multipoll [35]	21	18	8	0.06s	5	$\surd$
Central Server Model (CSM)[3]	13	14	4	0.03s	5	$\surd$
Readers-Writers [3]	7	7	2	0.02s	4	$\surd$
2x2 Mesh [3]	32	32	8	0.07s	5	$\surd$

Figure 4.4: Profile of the case-studies: T=number of transitions; P=number of places; SF(size of $\mathcal{F}$)=number of minimal T-semiflows; ET=CPU execution time to compute $\mathcal{F}$ using GSPN on a Pentium 133Mhz; WS=size of the sliding window; I?=The Fundamental set is integral.

Applicability. To make the tests more interesting, we considered models of concurrent and productions systems taken from [3, 18, 35]. Furthermore, in order to study the scalability of our approach we restricted ourselves to consider systems with *parametric* initial markings, where the parameter is the number of *initial tokens* in some given places of the net. For these examples, we were interested in computing the set of reachable states. As shown in Figure 4.4, most of the examples in [3, 18, 35] with the previous characteristics turned out to have *integral* fundamental set. We computed $\mathcal{F}$ using the structural library of GSPN within negligible execution times (see again Figure 4.4). Finally, we remark that only the Kanban system is a *free choice* net, all the other examples heavily rely on the use of *semaphores*.

Quality and Efficiency. In order to test the *quality* and *efficiency* of our heuristics, we compared the execution times of our prototype with those of GreatSPN [12], one of the more efficient tools for the generation of the reachability graph of a PN. We performed all experiments on a Pentium with a clock speed of 133Mhz, RAM memory of 32Mbytes, and swap area of 34Mbytes, allocating a priori 55Mbytes of memory to store the reachability set. The table in Figure 4.5 summarizes the results of a first serie of experiments. Surprisingly, the algorithm returned an *exact* representation (without redundancies) of the reachability set in all the examples (and different values for the parameter=number of tokens in the initial marking). In all the experiments of Figure 4.5 we never had to exploit the circularity of our memory organization: 55Mbytes where enough to store the reachability set. The cost of our heuristics and of the localized search turned out to be comparable to that of the efficient search of GreatSPN (despite the fact that GreatSPN makes also use of simplification rules). However, on examples like Reader-Writers GreatSPN was not able to compute the reachability graph for nets with more than 62 tokens in the initial marking (as indicated by the overflow flag $\surd$). This fact is due to the overhead of a more sophisticated

CASE-STUDY	NT	ET-**Prot**	NS-**Prot**	CF	ET-**GSPN**	NS-**GSPN**
Kanban	2	1.530s	4,600	true	0.860s	4600
	4	229.070s	454,475	true	158.700s	454,475
	5	1464.270s	2,546,432	true	$\surd$	$\surd$
	6	$\surd$	$\surd$	--		
FMS	2	1.270s	3,444	true	0.460s	3,444
	4	249.170s	438,600	true	117.770s	438,600
	5	$\surd$	$\surd$	--	$\surd$	$\surd$
Multipoll	2	5.210s	11,328	true	2.190s	11,328
	4	56.280s	106,280	true	27.030s	106,280
	9	1164.750s	1,943,160	true	$\surd$	$\surd$
	10	$\surd$	$\surd$	--		
Mesh	2	178.870s	200,544	true	46.150s	200,544
	3	$\surd$	$\surd$	--	$\surd$	$\surd$
CSM	2	0.020s	76	true	0.010s	76
	32	23.180s	95,876	true	27.920s	95,876
	75	311.530s	1,170,704	true	538.450s	1,170,704
	115	1156.240s	4,162,544	true	$\surd$	$\surd$
	116	$\surd$	$\surd$	--		
Reader-Writers	4	0.030s	90	true	0.010s	90
	32	7.170s	64,889	true	10.250s	64,889
	62	94.350s	762,384	true	175.300s	762,384
	114	1069.020s	7,927,295	true	$\surd$	$\surd$
	115	$\surd$	$\surd$	--		

Figure 4.5: First serie of experimental evaluations: NT=Number of Tokens in the initial marking; ET=CPU Execution Time on a Pentium 133Mhz; NS=Number of reachable States; CF=value of the Complete Flag when PS returns; $\surd$=memory overflow; -**Prot**=executed on our prototype; -**GSPN**=executed on GreatSPN [12].

encoding of markings and to the organization of visited markings as a tree structure [9] (trade-off between efficient search operations and memory requirements). (Note: both our prototype and GreatSPN store the edges of the reachability graph on disk).

Scalability. In order to test the scalability of our method, we performed a second serie of experiments in which we successively reduced the quantity of memory allocated for storing the reachability set. The aim was to test the efficacy of the circular implementation of the memory. The results were quite surprising. For instance, as shown in Figure 4.6 we were able to scale up to 255 the number of tokens in the initial marking of Readers-Writers. In this case the net has approximatively 185 millions of reachable states. They would require approximatively 1300 Mbytes of memory to be stored. We were able to run the example

Readers-Writers (No. trans. m= 7) executed on our **prototype**								
NT	TM	NS	MS	NR	R	ET	CF	OF
255	45Mb	185,977,536	1,302Mb	28	96%	27,981s	true	
255	35Mb	185,977,536	1,302Mb	37	96%	27,996s	true	
255	25Mb	185,977,536	1,302Mb	52	98%	27,991s	true	
255	21Mb	66,252,650	463Mb	22	95%	9,719s	true	$\surd$
128	45Mb	12,440,544	87.1Mb	1	50%	1,723s	true	
128	35Mb	12,440,544	87.1Mb	2	67%	1,722s	true	
128	25Mb	12,440,544	87.1Mb	3	75%	1,721s	true	
128	15Mb	12,440,544	87.1Mb	5	85%	1,722s	true	
128	5Mb	12,440,544	87.1Mb	17	93%	1,723s	true	
128	3Mb	5,631,404	40Mb	13	92%	766.6s	true	$\surd$
64	35Mb	860,145	6Mb	0	100%	108s	true	
64	5Mb	860,145	6Mb	1	50%	108.1s	true	
64	1Mb	860,145	6Mb	6	86%	108.3s	true	
64	0.5Mb	860,145	6Mb	12	92%	108.5s	true	
64	250Kb	169,728	1.2Mb	4	80%	19.6s	true	$\surd$
32	0.4Mb	64,889	455Kb	1	50%	7.38s	true	
32	0.3Mb	64,889	455Kb	1	50%	7.33s	true	
32	75Kb	64,889	455Kb	6	86%	7.38s	true	
32	50Kb	23,099	162Kb	3	75%	2.38s	true	$\surd$

Figure 4.6: Second serie of experimental evaluations: NT=number of tokens in the initial marking; TM=total allocated memory; NS=number of reachable states; MS=NS*m; NR=number of rounds in the circular memory; R=(1-(TM/MS)(%) (saving ratio); ET=CPU execution time on a Pentium, 133Mhz; CF=complete flag; OF=overflow flag.

using *only* 25 Mbytes of memory, hence using 98% less memory space. The memory manager returned an overflow exception (indicated again with $\surd$) when we tried to use 21Mb of memory. Furthermore, for an initial marking with 128, 64, and 32 tokens we were able to compute the reachability set using (approximatively) 94% (TM=5Mbytes), 92% (TM=0.5Mbytes), and 84% (TM=75Kb) less memory space, respectively. We obtained similar results for the CSM example. The results on the other examples were less appealing, though we also managed to scale up FMS to an initial marking with 5 tokens.

4.5 Related Works

As mentioned in the introduction, structural techniques are traditionally used to compute over-approximations of the reachability set, see e.g. [53]. In [27], *traps* are used to improve the quality of the approximation. *Place* invariants can also be used to over-approximate the reachability set. Place invariants are the dual notion of T-semiflows, i.e., the solution of the system $y \cdot \mathbf{C} = 0$. Let $\mathbf{P}$ be the matrix of minimal P-semiflows. As shown in [53], the solution of the equation $\mathbf{P} \cdot x = \mathbf{P} \cdot m_0$ over-approximates the set of solutions of the state equation $m = m_0 + \mathbf{C} \cdot \sigma$, i.e., over-approximates the reachability set. Contrary, in our approach we have used T-semiflows to find under-approximations (useful for debugging) and to derive conditions to establish the quality of the approximation. Furthermore, differently from [27, 53], our approach is incorporated in state enumeration. We are not aware of other approaches where T-semiflows are used for under-approximating the reachability set. In [39], Miner and Ciardo use MDDs (Multi-valued Decision Diagrams) to store the reachability set; whereas, Pastor et al. [42] use P-invariants (semiflows) to improve a BDD-based encoding of the reachability set. Other compact data structures (like Sharing Trees) are tested on reachability problems of bounded PNs in [30, 54]. As mentioned in the introduction, our heuristics could be incorporated, e.g., in a BDD-based framework. Our use of heuristics shares some similarities with *depth-first* search algorithms [33, 58] for state enumeration, an approach used to compute an under-approximation of the *reachability graph*. In fact, our heuristics gives us conditions to detect acyclic paths of the reachability graph that go from the initial marking to the current marking. However, note that the use of Parikh vectors allows us to check the absence of cycles on *collections* of paths (all paths and related subpaths represented by the vector). Furthermore, the use of the second heuristics allows us to obtain more accurate information w.r.t. a generic depth-first search where only the current paths is memoized.

Chapter 5

A Distributed Version

The construction of the TRS requires the manipulation of an irregular, complex state space, whose growht is tipically unpredictable and not uniform. Hence, this part of the GSPN solution does not exhibit the regularity in data structure required for easy data parallel implementation nor the natural decoupling leading to straightforward message passing solutions [8].

In the message passing programming model the parallel architecture seen by the programmer is not the whole machine, but rather the single processing node. The code runs in multiple copies on the individual processors. Moreover, the message exchange among the processors is not transparent to the user, requiring explicit calls to send and receive primitives in the program. We decided to adopt the message passing paradigm as implemented by the standard MPI programming environment.

5.1 The Distributed Algorithm

There are two key factors in the distributed generation of the TRS: the first one is to uniform partition of the state space; the second one is the amount of communications between processors. With respect to the state space partitioning, exploiting properties 3.1 and 3.2, described in chapter 3. In particular we propose a distributed algorithm where all the states belonging to a level L are stored in the memory of processor P using the following formula:

$$P = L\ MOD\ TotProc$$

Where $TotProc$ is the amount of available processors. The algorithm can be sketched as shown in Figures 5.1, 5.2 and 5.3.

The algorithm is organized as follows. After an inicialization phase in which each processor (identified by the *rank* value ranging between $0 \ldots (ToTMachines - 1)$, according to the MPI terminology) obtain a local copy of the static elements of the PN model, the first processor (namely the one with rank 0) generates new states starting from M_0 and send them to the next machine (the one identified with rank 1) . The machine identified with rank 1 receives the states generated by machine 0 stores them, and generates new states that are sent to machine 2. The process is repeated for each pair of contiguous machines. The last machine (the one identified by rank $(TotMachines - 1)$) send its new generated states to the first machine (rank 0).

We achieve a reduction in the amount of communications overhead between two processors by grouping many states into a single message. This buffering technique has a great impact in the performance of our algorithm as we have measured in our experiments.

5.2 Experimental Results

In the same way as we developed the experiments for the sequential version, we verified first of all the correctness of our distributed algorithm by checking that it produced the same reachability graph structure as the sequential RG builder. Subsequently we started to compare the performance of the various RG builders both on terms of CPU time and of memory requirements on the same machine. Our chosen performance measure in the *throughput*, defined as:

$$\frac{\text{number of states inserted into the RG}}{\text{elapsed time}}$$

Memory constraints are reflected both as a slow-down of the computation process due to the cache miss or pagination overhead, and as a maximal number of states that can be generated using a given platform.

We believe that with the current trend of low-cost computer technology evolution the main limitation is the memory requirement, that is limiting the maximum size of the RG that can be generated, rather than CPU time to complete the execution in the cases in which the graph is small enough to fit memory requirements. We ran our experiments on two different PC cluster platforms (Table 5.1) with similar Linux operatin system, in order to study the effects of CPU speed, memory size, and communication network bandwith. In both platforms we used from 2 to 6 processors.

Algorithm ReachG($\mathcal{N}, M_0, T_i$)
$\mathcal{N}$: a bounded P/T net with $\mathcal{N} =< P, T, Pre, Post, Inh >$
M_0: Initial Marking;
T_i: the base of *t-invariant*;
$\sigma \leftarrow [0, 0, \ldots, 0]$: current firing count vector;
$NM \leftarrow [\sigma]$: an ordered list of markings;
$RS \leftarrow [\sigma]$: reachability set of the net;
$RG \leftarrow \emptyset$: reachability graph of the net;
$TotMachines$: Amount of available machines;
$ToSend \leftarrow \emptyset$: data to send to next machine;
$ToRec$: data received from another machine;
$MyId$: Identification Number for each machine $0 \ldots TotMachines - 1$;
$From$: Identification Number of a sending machine;
$List_i \leftarrow T_i$;
if ($MyId = 0$)
    $NextMachine \leftarrow 1$;
    $M \leftarrow M_0 + C \cdot \sigma$;
    for (each transition t_i enabled in M)
        $\sigma' \leftarrow \sigma$;
        $\sigma'[i] \leftarrow \sigma'[i] + 1$;
        $\sigma' \leftarrow Normalize(\sigma', List_i, T_i)$;
        $ToSend \leftarrow ToSend \cup \langle \sigma', \sigma, t_i \rangle$;
    endfor
    Send($NextMachine$,$ToSend$,"New");
endif
while()
    Receive($From$,$ToRec$,$Code$);
    if (Code = "New")
        AddGenerate();
    endif
    if (Code = "Old")
        SearchOld();
    endif
    if (Code = "End")
        exit;
    endif
endwhile

Figure 5.1: The Distributed Algorithm.

```
AddGenerate()
ToRec: Data received from another machine;
ToSend ← ∅: new states to send;
ToSendOld ← ∅: old states to send;
while (ToRec ≠ ∅)
    let ⟨σ, σ', t_i⟩ be the first element of ToRec;
    remove ⟨σ, σ', t_i⟩ from ToRec;
    if (σ' ∉ RS)
        insert σ' at the end of NM;
        RS ← RS ∪ σ';
    endif
    RG ← RG ∪ (σ --t_i--> σ');
endwhile
while (NM ≠ ∅)
    let σ be the first element of NM;
    remove σ from NM;
    M ← M_0 + C · σ;
    for ( each transition t_i enabled in M )
        σ' ← σ;
        σ'[i] ← σ'[i] + 1;
        σ' ← Normalize(σ', List_i, T_i);
        if (Level(σ') = Current() + 1)
            ToSend ← ToSend ∪ ⟨σ', σ, t_i⟩;
        else
            ToSendOld ← ToSendOld ∪ ⟨σ', σ, t_i⟩;
        endif
    endfor
endwhile
Send(ToSendOld,"Old");
if (ToSend ≠ ∅)
    NextMachine ← (Current() + 1) MOD TotMachines;
    Send(NextMachine,ToSend,"New");
else
    Broadcast("End");
    exit;
endif
```

Figure 5.2: Function AddGenerate().

SearchOld()
$ToRec$: Data received from another machine;
while $(ToRec \neq \emptyset)$
    let $\langle \sigma, \sigma', t_i \rangle$ be the first element of $ToRec$;
    remove $\langle \sigma, \sigma', t_i \rangle$ from $ToRec$;
    if $(\sigma' \notin RS)$
        insert σ' at the end of NM;
        $RS \leftarrow RS \cup \sigma'$;
        $RG \leftarrow RG \cup (\sigma \xrightarrow{t_i} \sigma')$;
    else
        $RG \leftarrow RG \cup (\sigma \xrightarrow{t_i} \sigma')$;
    endif
endwhile

Figure 5.3: Function SearchOld().

Plattform	Processor	Memory	Connection	Library
I	133 MHz Pentium	60+60 Mb	100 MBp/s Ethernet	MPICH
II	800 MHz Athlon	256+256 Mb	1 GB/s Ethernet	MPICH+GAMMA

Table 5.1: Distributed Platforms for the Experiments.

For our experiments we have used the same models already used for the sequential experiments described in chapter 3:

A - modified version of the Readers & Writers System [3], obtained by making all transitions timed;

B - Readers & Writers System with priorities [3];

C - modified version of the Central Server Model [3], obtained by making all transitions timed;

D - original version of the Central Server Model [3] (with priority specification);

E - modified version of the Kanban Model with Failures and Repairs [3] in the particular case of 4 stages, obtained by making all transitions timed;

F - original version of the Kanban Model with Failures and Repairs [3] in the particular case of 4 stages (with priority specification);

G - Multipoll System [35] (with priority specification).

In the same way as we did in the experiments performed to validate our sequential algorithm, we ran the distributed version the models from low values of their marking parameter K until our prototype could not complete the task because of lack of memory. Each run was performed varying the amount of available processors.

Let us first discuss the measurement methodology that we followed. In Figure 5.4 (a) we report the actual CPU times measured when repeating 100 times the construction of the Reachability Graph for model A with parameter $K = 64$ using five processors on the platforms I and II. In each case we performed 100 runs, and results are shown in figure 5.4. As showed in the histograms (5.4 (b) and (d)) we have an higher data dispersion on platform I than platform III. A reason for this behaviour is the fact that platform I is used by other users, despite we conducted the experiments in periods of very low utilization. Instead platform III utilizes a close net.

For the experiments developed on platform I we decided to perform 30 replicas for each experiment, eliminate the two best and worst results and computing the average of the remaining 26 values.

On platform II we did 10 replications for each experiment, eliminated the best and the worst values and computed the average of the remaining 8 values.

5.2.1 Experimental Results for Model without Priority

In this section we present the results obtained with the models A, C and E. that is the models without priority specification.

5.2.1.1 Experimental Results on Platform I

The results obtained on this platform are showed in Table 5.2 and Figure 5.5. In Figure 5.5 we show the throughput obtained by the GreatSPN, our sequential algorithm (presented in chapter 3) and our distributed distributed algorithm varying the amount of processor from 2 to 6. In Figure 5.5 we can see that for low values of K the sequential versions have a better performance that decrease when the value of K increases. Between the different configurations of our distributed algorithm we can see that the lower performance is achieved for those configurations with the higher amount of processor (5 and 6). The higher performance correspond to the configuration with two and three processors. A first answer for this behaviour is the fact that MPI performance depends on the amount of working processors. A second answer is given for the behaviour of the algorithm when a processor generates an state whose level is lower than the current one. As we showed in

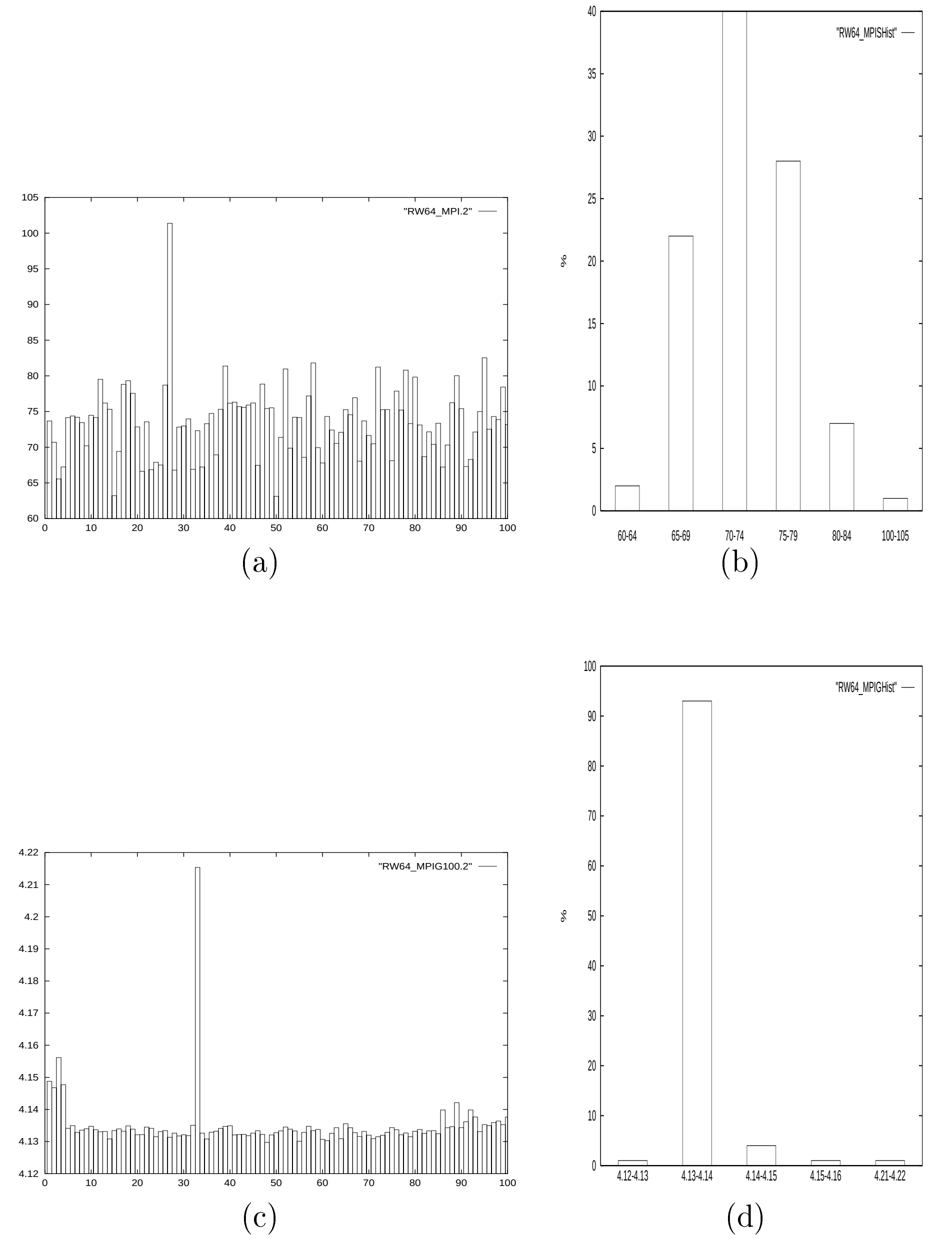


Figure 5.4: Variability of elapsed time when repeating experiments; (a) 100 replications on Platform I, (b) its histogram, (c) 100 replications on Platform II, (d) its histogram

PN Model			Processors on Platform I				
	K	$\|\|RG\|\|$	2	3	4	5	6
A (Readers & Writers)	2	19	0.03	0.55	0.16	0.39	0.33
	4	90	0.08	0.58	0.29	0.43	3.66
	8	615	0.17	0.62	0.38	0.52	3.78
	16	5,661	0.23	1.53	0.58	0.76	4.1
	32	64,889	1.72	2.02	2.45	9.34	11.57
	64	869,145	26.74	30.12	37.65	63.6	69.8
	128	12,440,545	319.35	300.36	408.58	854.89	868.25

Table 5.2: Results for Readers & Writers Model without priorities on platform I.

chapter 3, those states that belong a previous level are associated to the closure of a path whose length is the sum of the non-null elements of a *t-semiflow*. The model used has two *t-semiflows* whose associated lenght is 4. This means that if we use two processors an existing state will be found in the other processor (as the same as the new states). With three processors the already existing states are found in the same processor this implies that no communication is needed and only one processor is working in that step.

5.2.1.2 Experimental Results on Platform II

Platform II uses MPICH atop GAMMA [15], [13], [14], [16]. The Genoa Active Message MAchine (GAMMA) is an attempt to obtain good performance out of a cluster of PCs by adopting an optimized communication protocol and a suitable abstraction of the interconnect.

The results obtained on this platform are showed in Table 5.3 and Figure 5.6. In the three experiments conducted we have increased the value of K until to saturate the amount of available memory. With the first model we can able to arrive to a value for $K = 263$ using 6 processors, this answer the fast decrease of the performance in this case is due to the fact that we need 2 bytes for representing values over 255. Also we note that the better performance is achieved by the the configurations with higher amount of processor (5 and 6), the worst perfomance corresponds to those with three processors by the fact of the associated *t-semiflows* of the model.

The performance of the Central Server Model is showed if Figure 5.6 (b). We can see that alway the better performance in achieved by the distributed configuration and as same as in the Readers & Writers model the better one are for 5 and 6 processors. The worst distributed performance correspond to the configuration with three processors, again the answer is related to the associated *t-semiflows* of the model.

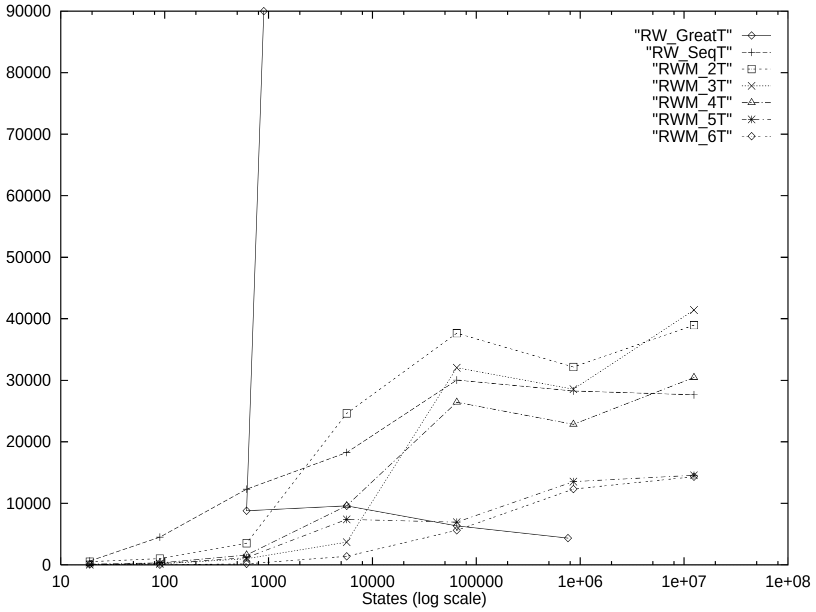


Figure 5.5: Throughput of Readers & Writers on Platform I

The results of the Fail Kanban Model with failures are less conclusive. As showed in Figure 5.6 (c) GreatSPN present an initial better performance than our distributed version. The answer for this behaviour is due to fact that this model is necesary to rearrange the states of a level during it generation process.

5.2.1.3 An Attempt to Reduce the Communications

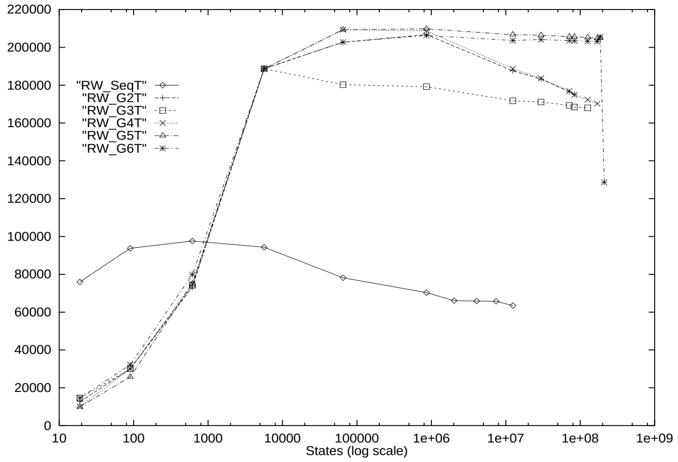
The analysis of the results on Platform I has moved us to develop a possible optimization of our algorithm which consist in to store a fix amount of levels in each execution step. The results of this optimization are showed in Figures 5.7 and 5.8.

As showed in Figure 5.7 the best performance is achieved with five and six processors, but this performance is not better than the previous approach. This behaviour is confirmed in the results showed in Figure 5.8 with a more efficient platform it is not possible to achieve a better performance.

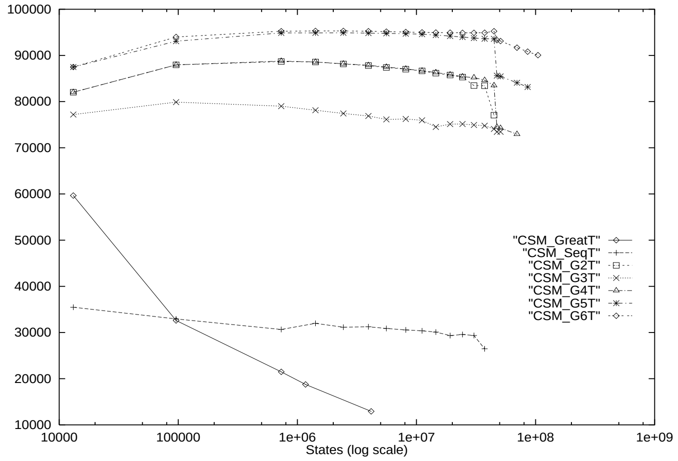
The reason of this behaviour is due to the buffering technique used to send the states from one processor to another. With this technique we obtain a degree of parallelism that explain the better performance with a larger amount of communications.

PN Model			Processors on Platform II				
	K	$\|RG\|$	2	3	4	5	6
A (Readers & Writers)	16	5,661	0.03	0.03	0.03	0.03	0.03
	32	64,889	0.32	0.36	0.31	0.31	0.32
	64	869,145	4.16	4.8	4.12	4.1	4.17
	128	12,440,545	66.31	72.44	65.92	60.18	61.09
	160	29,746,521	162.31	173.81	161.96	144.12	145.72
	200	71,412,151	404.56	421.72	403.74	347.17	350.79
	208	83,325,165	475.96	494.71	476.29	405.27	409.48
	231	125,936,966	m.o.	749.6	730.65	613.5	619.46
	249	169,299,397	m.o.	m.o.	994.33	827.03	833.63
	255	185,977,536	m.o.	m.o.	m.o.	907.05	904.83
	263	210,095,710	m.o.	m.o.	m.o.	m.o.	1,633.46
C (Central Server)	16	13,124	0.16	0.17	0.16	0.15	0.15
	32	95,786	1.09	1.2	1.09	1.03	1.02
	64	732,420	8.26	9.27	8.25.	7.72	7.69
	128	5,724,675	65.57	75.42	65.45	60.28	59.36
	240	37,327,044	447.36	499.13	441.02	398.8	393.25
	255	44,739,584	580.56	603.97	535.9	478.05	469.72
	260	47,412,564	m.o.	645.57	636.55	553.75	508.08
	266	50,758,124	m.o.	691.04	683.6	593.94	545.08
	296	69,861,924	m.o.	m.o.	957.74	830.93	762.08
	317	85,753,576	m.o.	m.o.	m.o.	1031.11	944.29
	339	104,811,121	m.o.	m.o.	m.o.	m.o.	1163.84
E (Kanban)	1	4,096	0.12	0.12	0.12	0.12	0.12
	2	104,976	3.13	3.16	3.11	3.07	3.13
	3	1,048,576	33.63	33.92	33.39	32.65	33.52
	4	6,259,000	210.79	212.66	209.42	202.92	209.86
	5	26,873,856	946.12	950.61	932.56	895.17	936.8

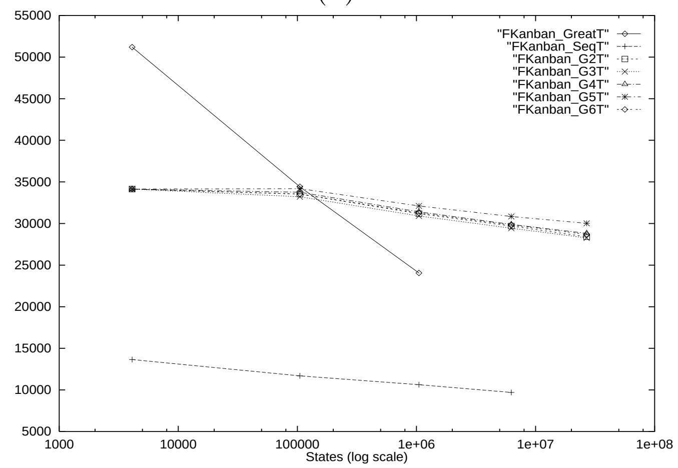
Table 5.3: Results on platform II for the models without priority. m.o. = memory overflow



(a)



(b)



(c)

Figure 5.6: Throughput for Models without priority on Plattform II: (a) Readers & Writers, (b) Central Server, (c) Kanban with Failures.

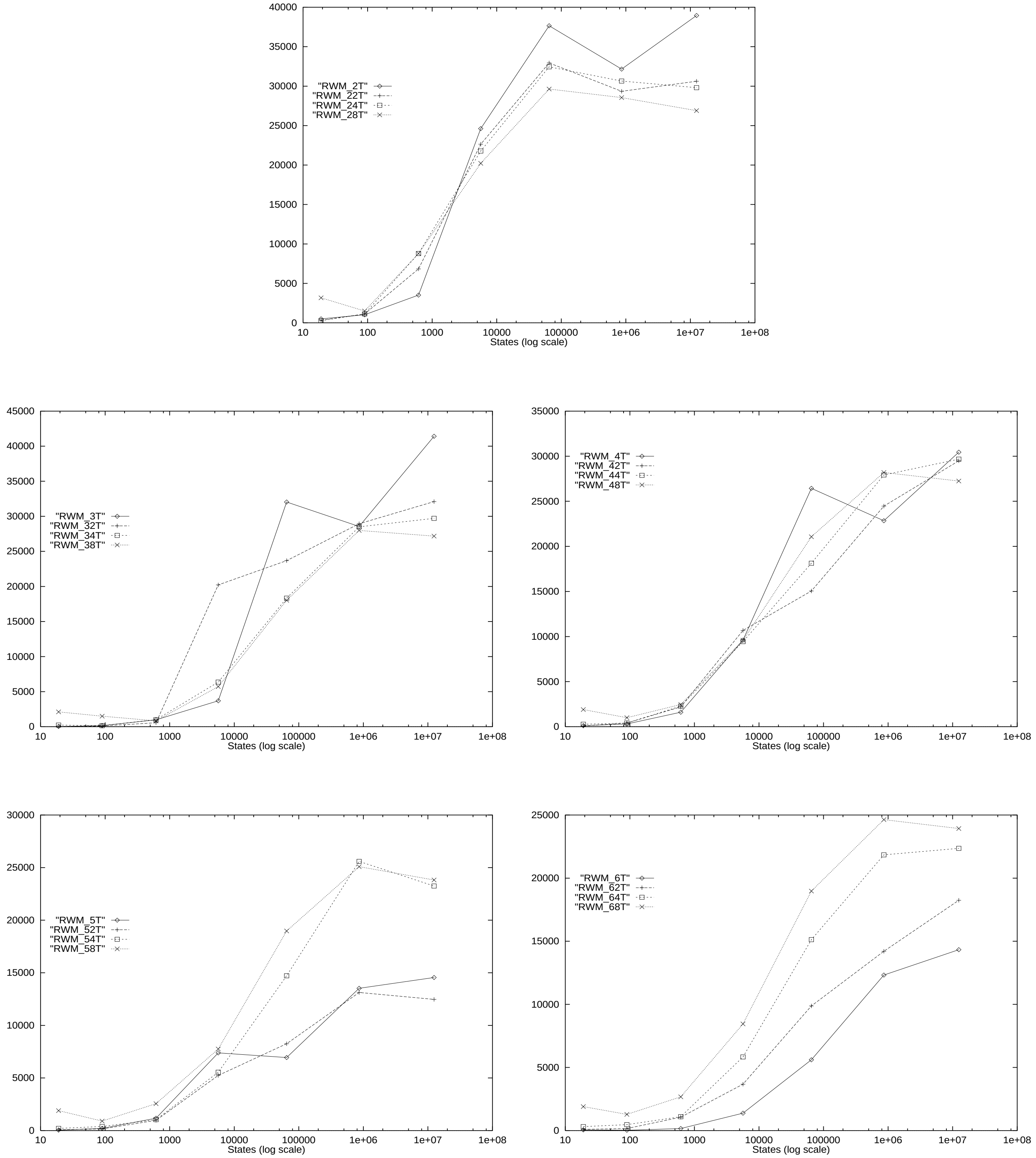


Figure 5.7: Throughput of Readers & Writers on Platform I with Variable Levels.

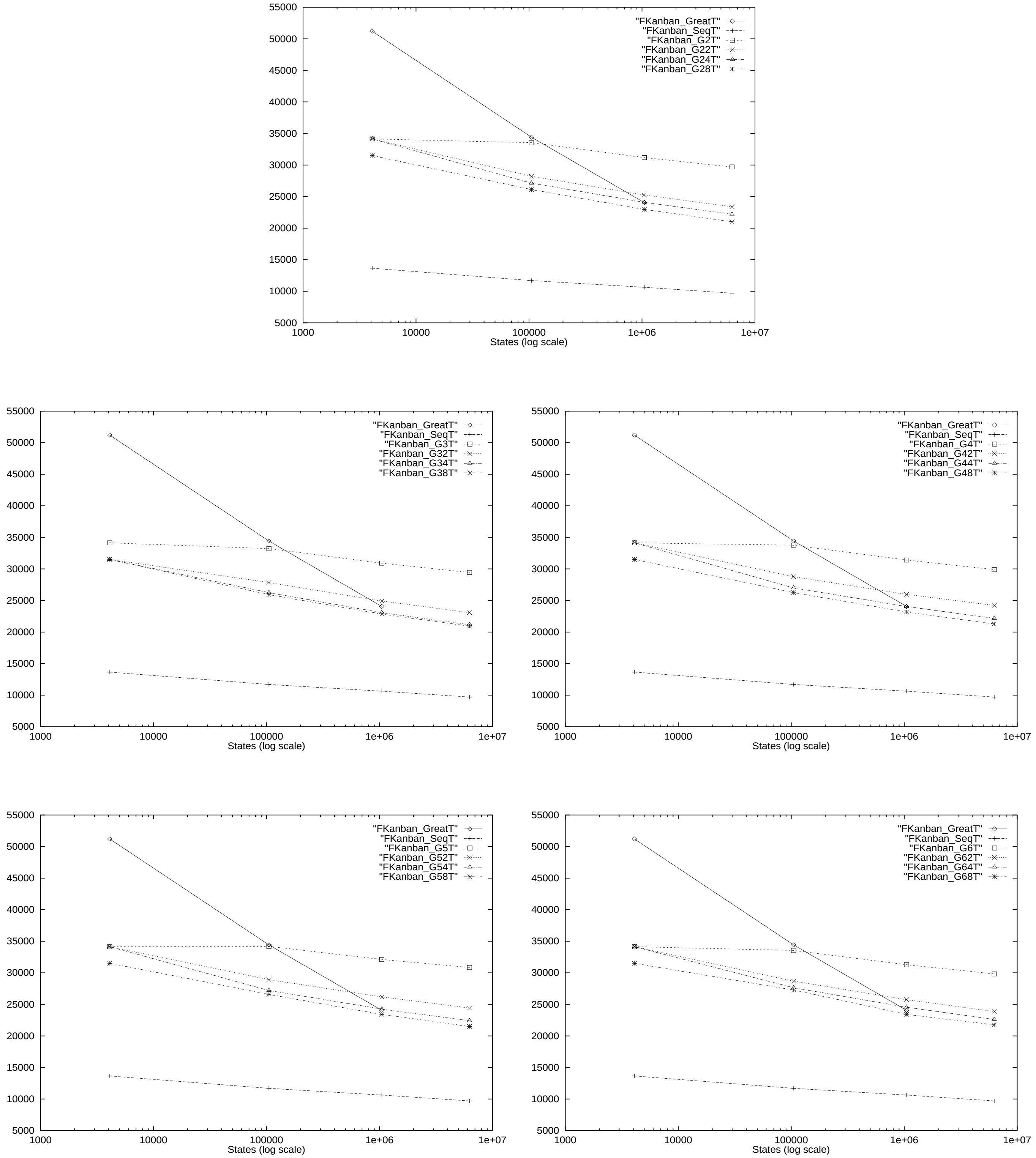
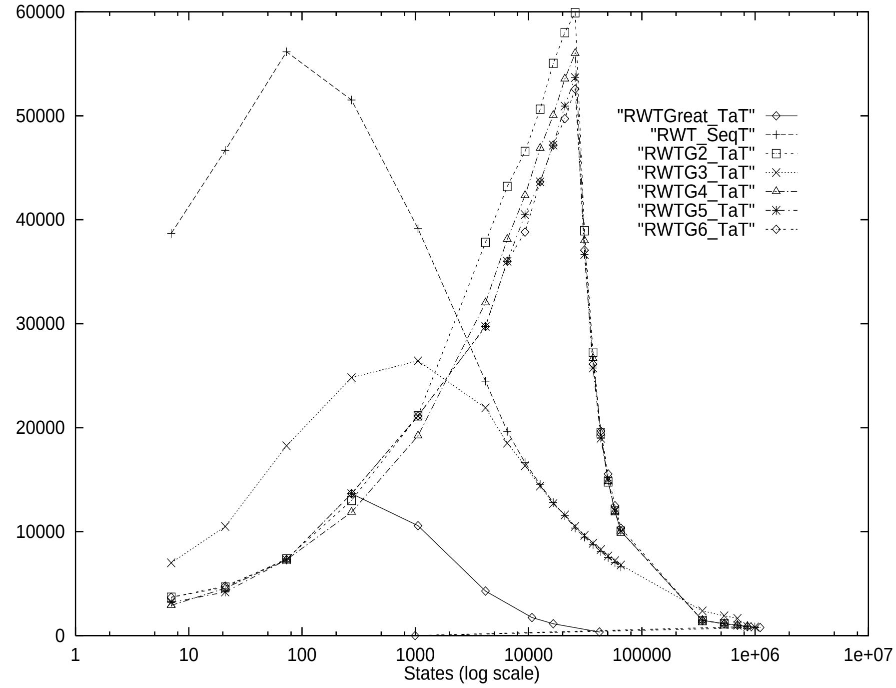


Figure 5.8: Throughput of Kanban Model with failures on Platform II with Variable Levels.

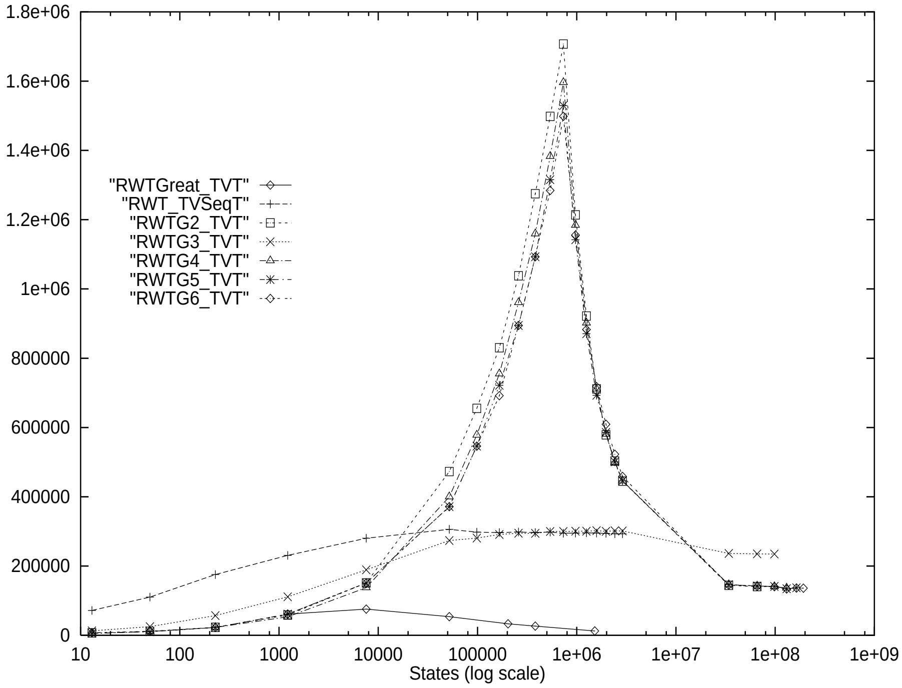
5.2.2 Experimental Results for Models with Priority

In the PN models with priority an increase of the interchange of messages is produced between the processors. This happens because there are states that belong to a lower level but there are not in the associated processor. In this situation the receiving processor send a warning message to the sender processor.

The results obtained on the platform II are showed in Table 5.4 and Figures 5.9,5.10, 5.11 and 5.12. In the four experiments conducted we have increased the value of K until to saturate the amount of available memory. In the four models analyzed the experiments conducted with our distributed algorithm presents a better perfomance than GreatSPN and our sequential version in spite of to need to do many more Send/Receive operations. Also we note that there is no significative diference in the throughput obtained with different configurations. The only exception is the last experiment with the Multipoll Model, because of the role of its *t-semiflows*.

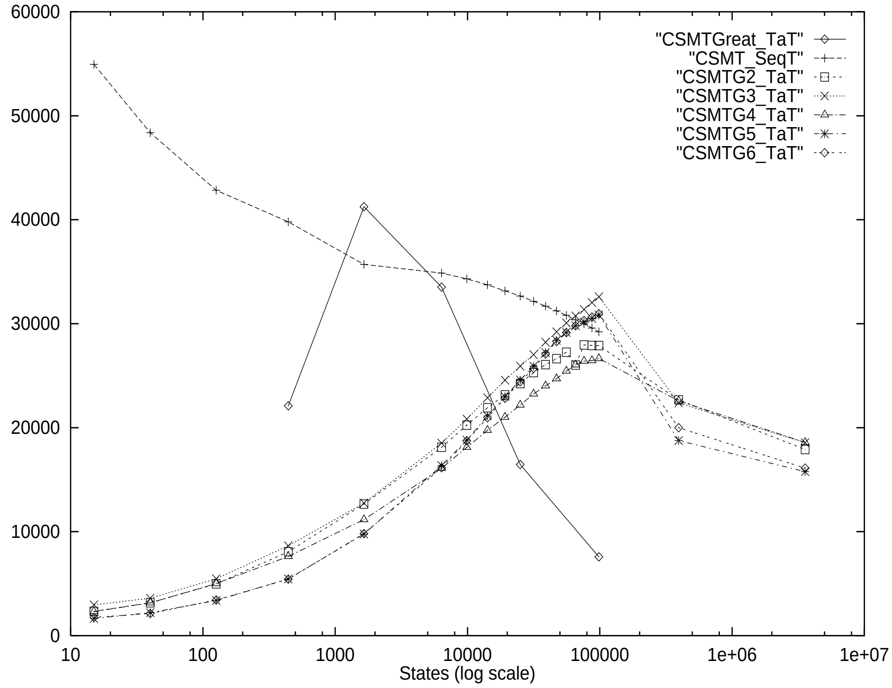


(a)

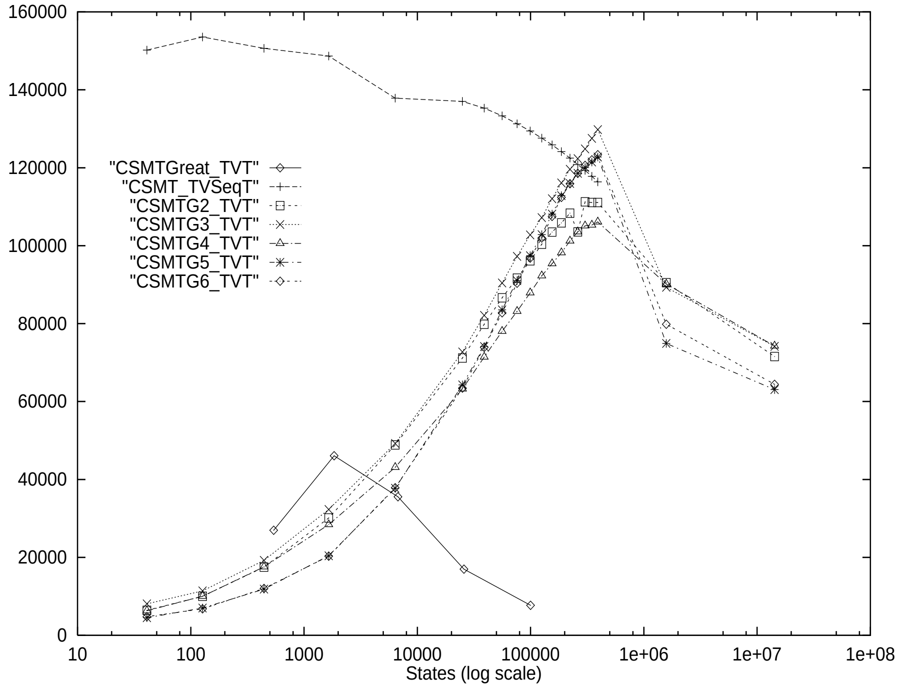


(b)

Figure 5.9: Throughput of Readers & Writers. (a) Tangible Throughput; (b) Overall Throughput.



(a)



(b)

Figure 5.10: Throughput of Central Server Model. (a) Tangible Throughput; (b) Overall Throughput.

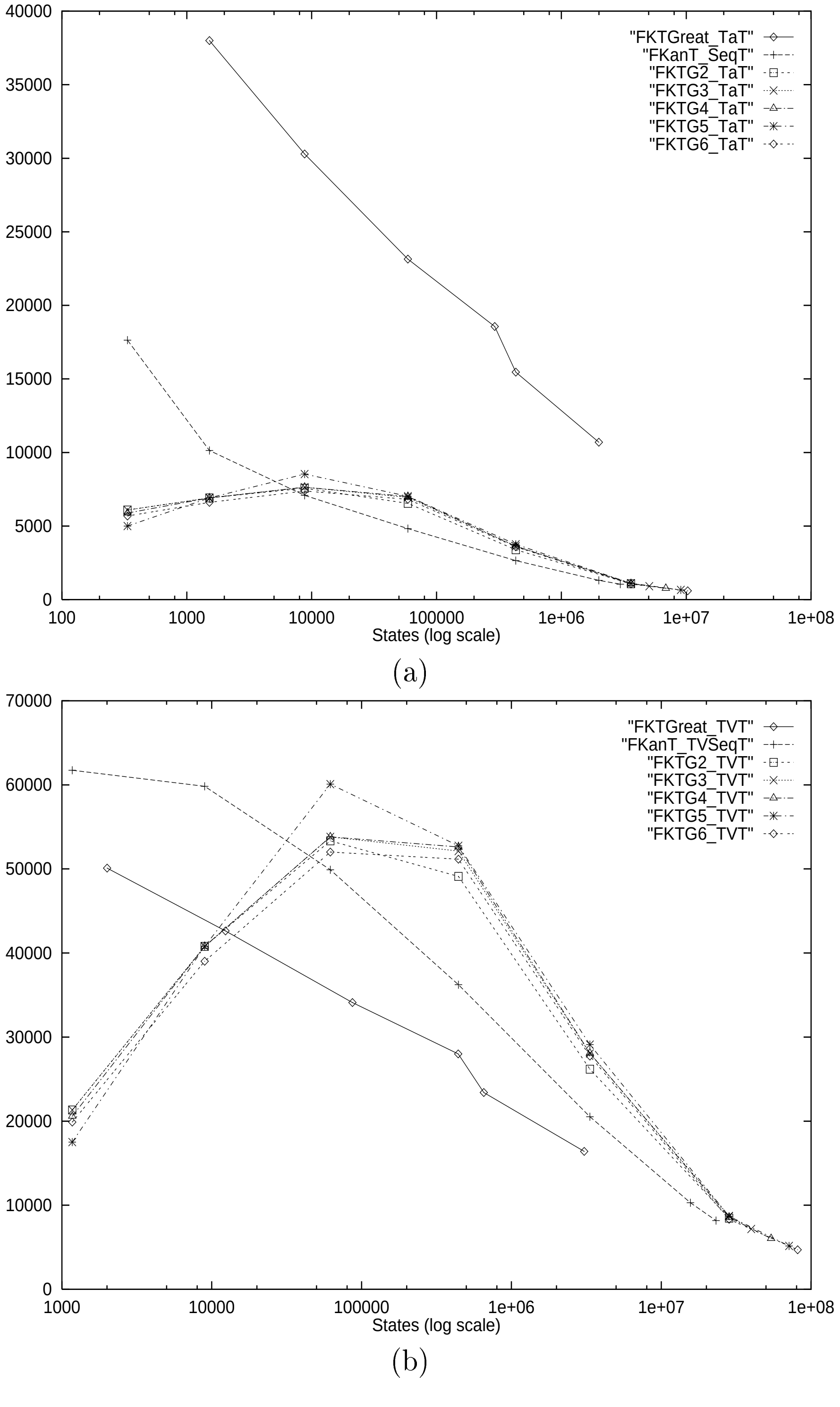
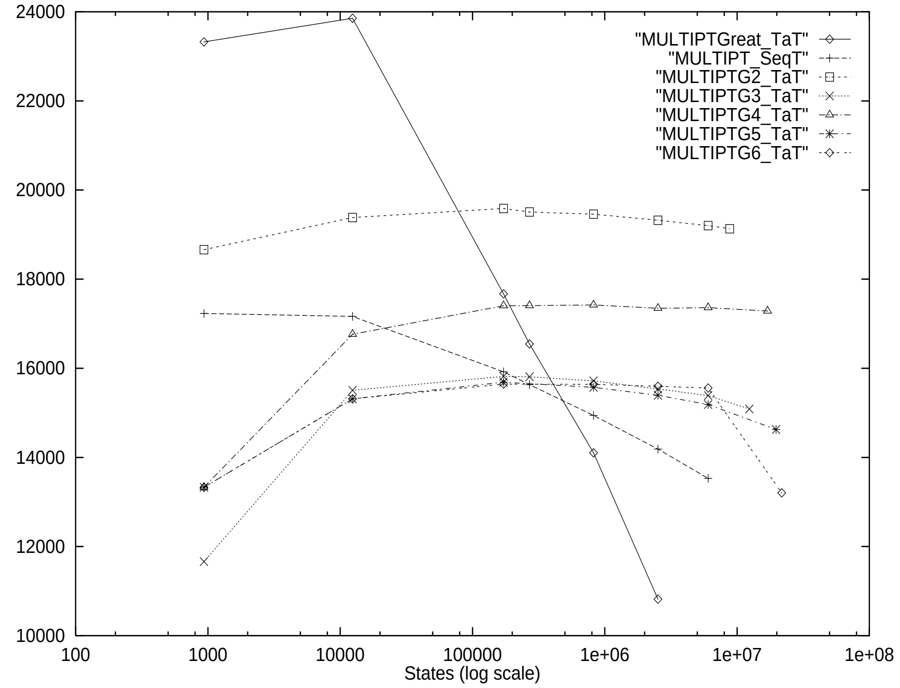
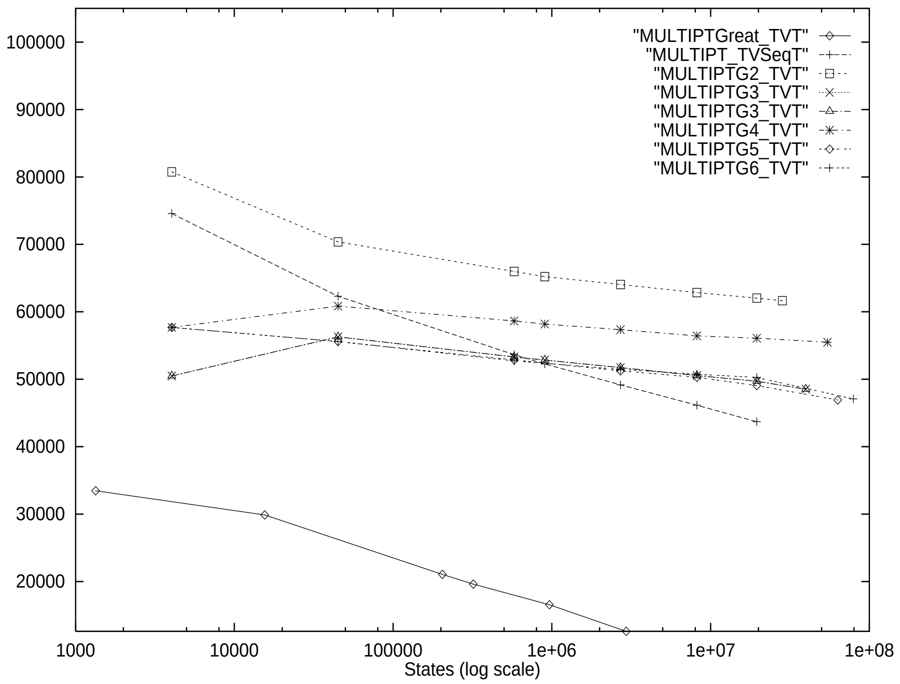


Figure 5.11: Throughput of Kanban Model. (a) Tangible Throughput; (b) Overall Throughput.



(a)



(b)

Figure 5.12: Throughput of Multipoll Model. (a) Tangible Throughput; (b) Overall Throughput.

PN Model				Processors on Platform II				
	K	$\|TRG\|$	$\|VRS\|$	2	3	4	5	6
B (R & W)	32	1,057	6,511	0.05	0.04	0.05	0.05	0.05
	64	4,161	47,839	0.11	0.19	0.13	0.14	0.14
	128	16,513	366,015	0.3	1.3	0.33	0.35	0.35
	255	65,281	2,828,799	6.5	9.6	6.49	6.49	6.31
	585	342,811	33,709,649	236.4	144.17	233.09	233.19	235.10
	730	533,631	65,369,674	467.88	280.18	463.23	462.65	466.68
	835	698,061	97,728,399	m.o.	419.39	699.74	697	701.81
	930	854,574	129,543,593	m.o.	m.o.	954.58	969.25	975.87
	996	994,975	163,234,526	m.o.	m.o.	m.o.	1,195.75	1,204.74
	1050	1,105,725	191,479,910	m.o.	m.o.	m.o.	m.o.	1417.21
D (Central Server)	32	1,650	4,721	0.13	0.12	0.14	0.16	0.16
	64	6,370	18,657	0.35	0.34	0.39	0.38	0.39
	128	25,026	74,177	1.03	0.96	1.12	1.01	1.02
	255	98,432	293,506	3.52	3.02	3.69	3.19	3.17
	512	395,011	1,181,441	17.42	17.65	17.49	21.05	19.75
	1540	3,562,793	10,677,591	199.03	192.08	191.79	225.88	221.15
E (Kanban)	4	8,784	53,099	1.16	1.15	1.15	1.03	1.19
	8	59,024	384,036	9.02	8.5	8.42	8.4	8.66
	16	431,736	2,907,424	127.58	119.05	118.86	114.72	120.36
	33	3,609,424	24,773,477	3,357.02	3,255.04	3,320.3	3,268	3,412.64
	45	9,044,407	62,290,698	m.o.	m.o.	m.o.	13,828.99	13,915.12
	47	10,288,860	71,030,208	m.o.	m.o.	m.o.	m.o.	17,284.26
F (Multipoll)	4	12,405	32,625	0.64	0.8	0.74	0.81	0.07
	8	171,765	407,025	8.77	10.86	9.87	10.95	10.98
	16	2,522,613	5,683,185	130.56	126.36	145.43	163.87	161.74
	20	6,047,349	13,487,985	314.93	393	348.31	398.19	388.72
	22	8,7945,33	19,543,905	459.69	n.a.	n.a.	n.a.	n.a.
	24	12,385,854	27,440,625	m.o.	820.89	n.a.	n.a.	n.a.
	26	16,977,647	35,751,985	m.o.	m.o.	982.28	n.a.	n.a.
	27	19,702,048	43,488,480	m.o.	m.o.	m.o.	1,346.98	n.a.
	29	21,766,650	57,537,000	m.o.	m.o.	m.o.	m.o.	1,684.32

Table 5.4: Results on platform II for the models with priority. m.o. = memory overflow; n.a. = not available.

Chapter 6

Conclusion and Future Work

We have described our algorithm for the construction of the RG using a state encoding based on the firing transitions and the application of some properties of the reachability graph. Our execution time results are comparable with the GreatSPN RG builder in case of nets without priorities and we largely surpassed the amount of states that can be kept in a given quantity of memory. Moreover our distributed version of the algorithm showed reasonably good speed on clusters with fast interconnections networks, allowing to further increase the total size of problems that can be solved without substantial speed penalties. We believe that the memory limitation is a more important factor affecting the usability of a tool compared to the CPU time to obtain the solution, as in any case the solution of solvable model is fairly quick on modern processors. Therefore we think our prototype can be already considered a good contribution to the state of the art from a practical point of view.

6.1 Achieved Results

This research has been undertaken with two main objetives in mind. Our first objetive was showing that the state encoding based on the firing transitions could be a valid alternative for the reachability graph generation. The second objetive was to develop an efficient distributed version.

We consider that we achieved both objetives because our state encoding has showed to be competitive in terms of memory resources and we were able to develop a distributed version of our algorith directly from the sequential version thanks to the properties associated to the reachability graph made evident with the encoding based on the firing transitions.

Another important results is the extension of our work in order to develop an algorithm for validating concurrent systems modeled as bounded Petri Nets. Our method is based on two structural heuristics based on the properties of T-semiflows. One of the main feature of this modified algorithm is that it allows us to organize main memory as a *circular array*, so as to re-use part of the slots allocated during the exploration of the state space.

6.2 Future Work

We must emphasize that we did not apply any reductions at the net level, and that the application of other techniques can further improve our results. We identified one possible improvement for the case of nets with priority that could perhaps make our algorithm always competitive compared to GreatSPN, even in case of independent subnets of immediate transitions. Another possible improvement is to investigate the use of implicit methods for the state encoding [17].

Appendix A

PN Models

In this appendix we present the PN modes used to carry out our experiments.

A.0.1 Models without Priorities

The models presented here are the next:

- Readers & Writers
- Central Server
- Kanban with Failures and Repairs (with 4 stages)
- Mesh 2x2
- Flexible Manufacturing System

A.0.2 GSPN Models

The models presented here are the next:

- Readers & Writers
- Central Server
- Kanban with Failures and Repairs (with 4 stages)
- Multipoll

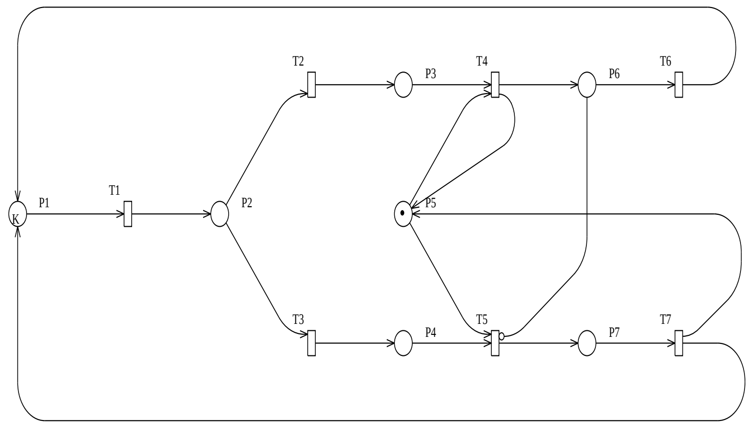


Figure A.1: Readers & Writers Model without priority

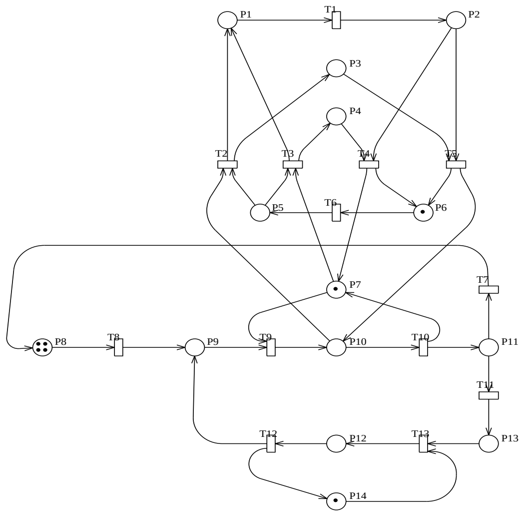


Figure A.2: Central Server Model without priority

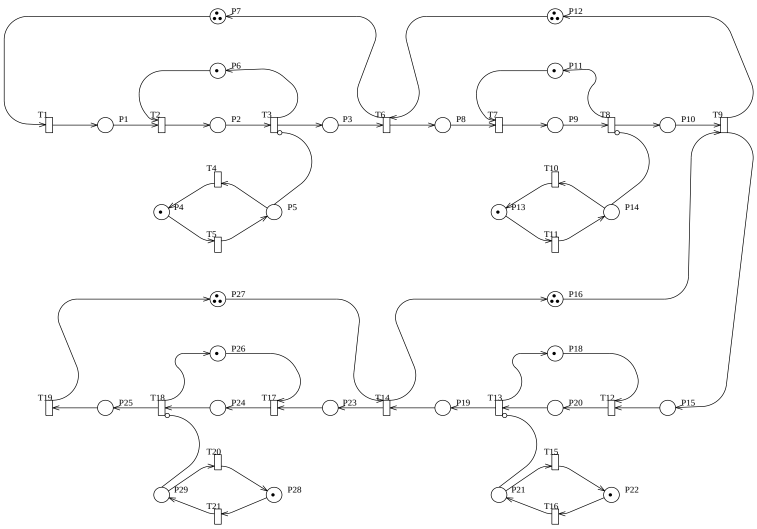


Figure A.3: Kanban Model with Failures without priority

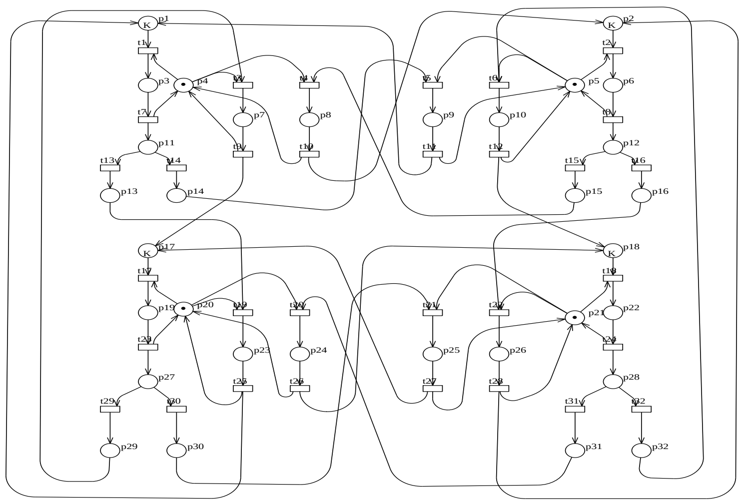


Figure A.4: Mesh 2x2 Model without priorities

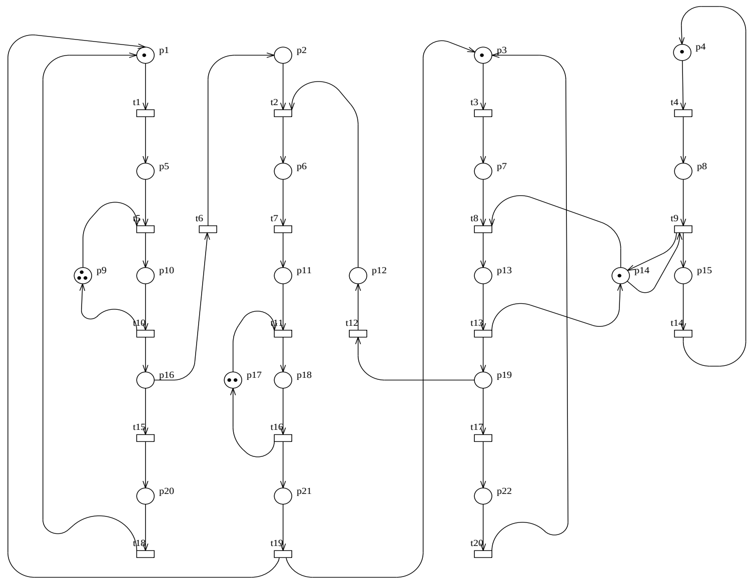


Figure A.5: Flexible Manufacturing Model without priorities

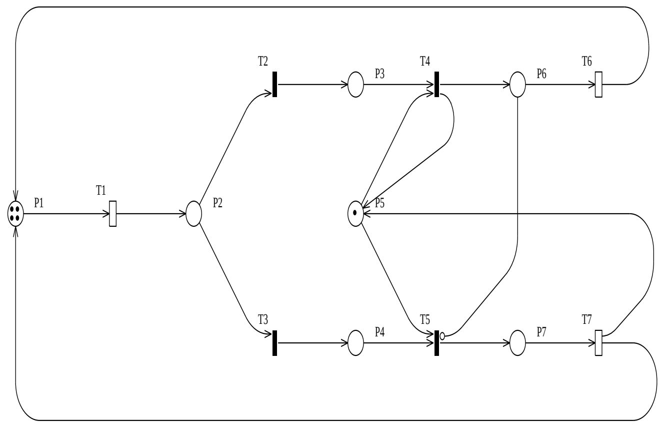


Figure A.6: Reader & Writers Model

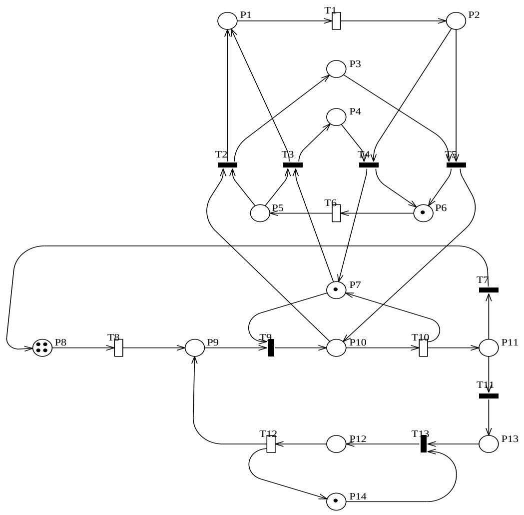


Figure A.7: Central Server Model

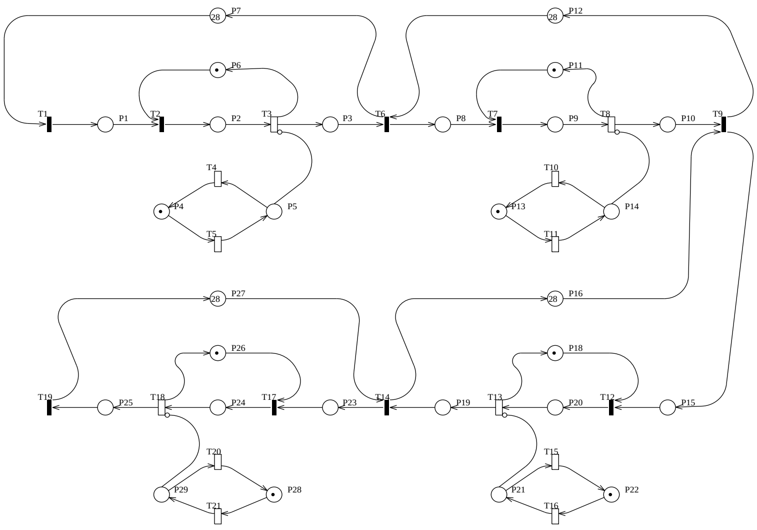


Figure A.8: Kanban Model with Failures

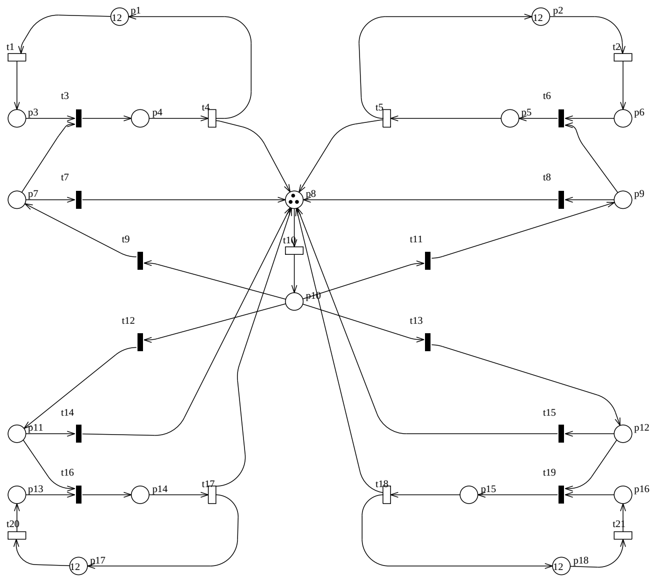


Figure A.9: Multipoll Model

Bibliography

[1] M. A. Marsan, S. Donatelli, F. Neri and U. Rubino. On the construction of abstract GSPNs: An exercise in modelling. In Proc. *IEEE Int. Workshop Petri Nets and Performance Models.* Melbourne, Australia, 1991.

[2] M. Ajmone Marsan, G. Balbo and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(2):93-122.

[3] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets.* John Wiley & Sons, 1995.

[4] S. C. Allmaier, M. Kowarschik and G. Horton. State Space Construction and Steady-state Solution of GSPN on a Shared-Memory Multiprocessor. In Proc. 7^{th} *Int. Workshop on Petri Nets and Performance Models.* Saint Malo, France, June 1997. IEEE-CS Press, 1997.

[5] R.E. Bryant. Binary Decision Diagrams and Beyond: Enabling Technologies for Formal Verification. In Proc. *Int. Conference on Computer-Aided Design, 1995.* IEEE-CS Press, 1995.

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. *In* Proc. LICS '90. 1990.

[7] R. Carvajal-Schiaffino, G. Delzanno and G. Chiola. Combining Structural and Enumerative Techniques for the Validation of Bounded Petri Nets. *Tools ans Algorithms for the Construction and Analysis of Systems, 2001.* Springer LNCS 2031, 2001.

[8] S. Caselli, G. Conte and P. Marenzoni. Parallel State Space Exploration for GSPN Models. *Application and Theory of Petri Nets, 1995.* Springer LNCS 935, 1995.

[9] G. Chiola Compiling Techniques for the Analysis of Stochastic Petri Nets. In Proc. 4^{th} *Int. Conf. on Modelling Techniques and Tools for Performance Evaluation*, 1989.

[10] G. Chiola and R. Carvajal-Schiaffino. A Reachability Graph Construction Algorithm Based on Canonical Firing Count Vectors. In Proc. 9^{th} *Int. Wokshop on Petri Nets and Performance Models.* Aachen, Germany, September 2001. IEEE-CS Press, 2001.

[11] G. Chiola, G. Franceschinis and R. Gaeta. Modelling Symmetric Computer Architectures by SWNs. *Application and Theory of Petri Nets, 1994.* Springer LNCS 815, 1994.

[12] G. Chiola, G. Franceschinis, R. Gaeta and M. Ribaudo. GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation,* Vol. 24, Nos. 1-2, pp. 47-68, Nov. 1995.

[13] G. Ciaccio. *A Communication System for Efficient Parallel Processing on Cluster of Personal Computers.* PhD thesis, Università di Genova, DISI-TH-1999-02, 1999.

[14] G. Ciaccio. *MPI/GAMMA home page.* http://www.disi.unige.it/project/gamma/mpigamma/

[15] G. Ciaccio, G. Chiola. *Genoa Active Message MAchine (GAMMA).* http://www.disi.unige.it/project/gamma/

[16] G. Ciaccio and G. Chiola. GAMMA and MPI/GAMMA on Gigabit Ethernet. In Proc. *7th EuroPVM-MPI.* Balatonfured, Hungary, September 2000. Springer LNCS 1908 , 2000.

[17] G. Ciardo. What a Structural World. In Proc. 9^{th} *Int. Wokshop on Petri Nets and Performance Models.* Aachen, Germany, September 2001. IEEE-CS Press, 2001.

[18] G. Ciardo, A. Miner. Storage alternatives for large structured state spaces. *Computer Performance Evaluation, 1997.* Springer LNCS 1245, 1997.

[19] J. M. Colom, E. Teruel and M. Silva Logical Properties of P/T systems and their analysis. *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques.* draft book for the Summer School Jaca (Spain) Vol I, 1998.

[20] D. D. Deavours, W. H. Sanders. An Efficient Disk-Based Tool for Solving Very Large Markov Models. *Computer Performance Evaluation, 1997.* Springer LNCS 1245, 1997.

[21] G. Delzanno, and J. F.Raskin. Symbolic Representation of Upward-closed Sets. *Tools and Algorithms for the Construction and Analysis of Systems, 2000.* Springer LNCS 1785, 2000.

[22] J. Desel. Basic Linear Algebraic Techniques for Place/Transition Nets. *In Reisig and Rozenberg [47]*, 1998.

[23] J.Desel and J. Esparza. *Free Choice Petri Nets.* Cambridge University Press, 1995.

[24] J. Desel and W. Reisig. Place/Transition Petri Nets. *In Reisig and Rozenberg [47]*, 1998.

[25] S. Donatelli. Superposed Generalized Stochastic Petri nets: definition and efficient solution. *Application and Theory of Petri Nets, 1994.* Springer LNCS 815, 1994.

[26] S. Donatelli. Tensor based Method in GSPN. *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques.* draft book for the Summer School Jaca (Spain) Vol II, 1998.

[27] J. Esparza and S. Melzer. Verification of safety properties using integer programming: Beyond the state equation. *Formal Methods in System Design*, 16:159-189, 2000.

[28] G. Florin, S. Natkin. Les reseaux de Petri stochastiques. *Technique et Science Informatiques*, 4(1):143-160.

[29] G. Franceschinis, L. Capra. Symmetries and exact lumping in SWN. *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques.* draft book for the Summer School Jaca (Spain) Vol II, 1998.

[30] F. Gagnon, J-Ch. Grégoire and D. Zampuniéris. Sharing Trees for 'On-the-fly' Verification. In Proc. 8^{th} *Int. IFIP Conference on Formal Description Techniques for Distributed Systems and Communications Protocols, FORTE'95*, 1995.

[31] B. Grahlmann. The PEP Tool. *In* Proc. CAV'97, Springer LNCS 1254, 1997.

[32] J. Hillston, M. Ribaudo. Stochastics Process Algebras. *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques.* draft book for the Summer School Jaca (Spain) Vol I, 1998.

[33] G. Holzmann. Algorithms for Automated Protocol Verification. *AT&T Technical Journal*, 69(2):32-44, 1988.

[34] C. Jard, Th. Jeron. Bounded-memory Algorithms. *In* Proc. CAV'91, Springer LNCS 575, 1991.

[35] P. Marenzoni, S. Caselli and G. Conte. Analysis of large GSPN Models: A Distributed Solution Tool. In Proc. 7^{th} *Int. Workshop on Petri Nets and Performance Models.* Saint Malo, France, June 1997. IEEE-CS Press, 1997.

[36] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* Kluwer Academic, 1993.

[37] P.M. Merlin, D.J. Farber. Recoverability of communication protocols: Implications of a theoretical study. *IEEE Transactions on Communications*, 24(9) 1036-1043.

[38] H. Miller, and S. Katz. Saving Space by Fully Exploiting Invisible Transitions. *In* Proc. of CAV '96, Springer LNCS 1102, 1996.

[39] A. Miner, G. Ciardo. Efficient Reachability Set Generation and Storage using Decision Diagrams. *Application and Theory of Petri Nets, 1999.* Springer LNCS 1639, 1999.

[40] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transaction on Computers*, 31(9):913-917.

[41] E. Pastor, O. Roig, J. Cortadella and R.M. Badia. Petri Net Analysis Using Boolean Manipulation. *Application and Theory of Petri Nets, 1994.* Springer LNCS 815, 1994.

[42] E. Pastor, J. Cortadella, M. A. Pena. Structural Methods to Improve the Symbolic Analysis of Petri Nets. *Application and Theory Petri Nets, 1999.* Springer LNCS 1639, 1999.

[43] C.A. Petri. *Kommunication mit Automaten.* Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[44] Petri Nets Tools Database. http://www.daimi.aau.dk/PetriNets/tools/db.html

[45] C. Ramchandani. *Analysis of Asynchronous Concurrent Systems by Timed petri Nets.* PhD thesis, MIT Cambridge MA, 1974.

[46] W. Reisig. *Petri Nets, An Introduction.* EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1982.

[47] W. Reisig and G. Rozenberg, editors. Lectures on Petri Nets I: Basic Models. *Advances in Petri Nets*, Springer LNCS 1491, 1998.

[48] O. Roig, J. Cortadella and E. Pastor. Verification of Asynchronous Circuits by BDD-based Model Checking of Petri Nets. *Application and Theory of Petri Nets, 1995.* Springer LNCS 935, 1995.

[49] W. H. Sanders, J. F. Meyer. Variable driven construction methods for stochastic activity networks. *Applied Mathematics and Performance/Reliability Models of Computer/Commnunication Systems*, 1987.

[50] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1994.

[51] J. Sifakis. Petri nets for performance evaluation. *In Proc. 3rd Int. Symp. IFIP*, 1978 75-93.

[52] M. Silva. *Las Redes de Petri en la Automática y la Informática.* Editorial Ac, Madrid, Spain, 1985. (in Spanish).

[53] M. Silva, E. Teruel, and J. M. Colom. Linear Algebraic and Linear Programming Techniques for Analysis of Place/Transition Net Systems. *In Reisig and Rozenberg [47]*, 1998.

[54] K. Strehl, L. Thiele. Interval Diagram Techniques For Symbolic Model Checking of Petri Nets. *In* Proc. DATE'99, 1999.

[55] F. Symons. *Modelling and Analysis of Communication Protocols Using Numerical Petri Nets.* PhD thesis, University of Essex, 1978.

[56] E. Teruel, G. Franceschinis and M. Silva. Untimed Petri Nets. *Performance Models for Discrete Event Systems with Synchronisations: Formalisms and Analysis Techniques.* draft book for the Summer School Jaca (Spain) Vol I, 1998.

[57] A. Valmari. Stubborn Sets for Reduced State Space Generation. *Advances in Petri Nets 1990.* Springer-Verlag LNCS 483, 1991.

[58] A. Valmari. The State Explosion Problem. *In Reisig and Rozenberg [47]*, 1998.

[59] G. Wimmel. A BDD-based Model Checker for the PEP Tool. *Technical Report*, University of Newcastle upon Tyne, 1997.